Existential Label Flow Inference via CFL Reachability*

Polyvios Pratikakis Michael Hicks Jeffrey S. Foster

July, 2005

Abstract

Label flow analysis is a fundamental static analysis problem with a wide variety of applications. Previous work by Mossin developed a polynomial time subtyping-based label flow inference that supports Hindley-Milner style polymorphism with polymorphic recursion. Rehof et al have developed an efficient $O(n^3)$ inference algorithm for Mossin's system based on context-free language (CFL) reachability. In this paper, we extend these results to a system that also supports existential polymorphism, which is important for precisely describing correlations among members of a structured type, even when values of that type are part of dynamic data structures. We first develop a provably sound checking system based on polymorphically-constrained types. As usual, we restrict universal quantification to the top level of a type, but existential quantification is first class, with subtyping allowed between existentials with the same binding structure. We then develop a CFL-based inference system. Programmers specify which positions in a type are existentially quantified, and the algorithm infers the constraints bound in the type, or rejects a program if the annotations are inconsistent.

1 Introduction

Label flow analysis is a program analysis that attempts to answer queries of the form "Does the value of expression e_1 flow to the value of expression e_2?" Answering such queries has a large variety of applications, including points-to analysis [2, 3], information flow [4], and type qualifier inference [5, 6].

Label flow analysis can be implemented as a *type-based program analysis* [1], in which static analysis is defined in terms of type inference. In type-based label flow analysis, as in other type-based analyses, context sensitivity for functions corresponds to parametric (universal) polymorphism. Mossin [7] gave the first polynomial time algorithm for subtyping-based label flow inference with Hindley-Milner style parametric polymorphism, including polymorphic recursion. Rehof et al [8, 26] showed how the label flow problem defined by Mossin could be reduced to the problem of context-free language (CFL) reachability. The resulting inference system runs in time $O(n^3)$, and follow-on work (some of which is cited above) has shown it to be quite efficient and useful in practice.

In this work, we develop a new extension to CFL-based label flow inference that supports *existential quantification* [9]. Intuitively, universal polymorphism allows multiple calls to the same function to be distinguished, avoiding some spurious flows. However, universal polymorphism aids

*University of Maryland, Computer Science Department Technical Report CS-TR-4007. This research was supported in part by NSF CCF-0346982 and CCF-0430118.

little in the analysis of data structures. For example, in label-flow based alias analysis, all elements of the same data structure usually become conflated, resulting in a "blob" of indistinguishable pointers that reduces precision [10]. With existential quantification, we can express correlations among members of a structured type, allowing us to model data structures much more precisely in many useful cases. For example, existentials can be used to correlate an integer length with a buffer having that length [11], to correlate data with the lock that must be held when accessing the data [12], or to correlate an environment with the closure that takes it as an argument [13]. Such correlations are particularly important when the identity of individual data structure elements cannot be discerned, e.g., when they are stored within a tree or list, or when they are generated based on dynamic information. In such a setting, the precision afforded by universal quantification is unavailable.

In the label flow systems presented in this paper, labels L are attached to types, as is standard. For example, int^L is the type of an integer with label L. We begin by presenting a sound subtyping-based label-flow system in the style of Mossin that uses *polymorphically constrained types* (Section 3). For example, the identity function can be given the type $\forall\alpha, \beta[\alpha \leq \beta].int^\alpha \rightarrow int^\beta$. The subtyping constraint $\alpha \leq \beta$ indicates that the label on the argument *flows* to the label on the result. The key technical novelty is that we also allow existential polymorphism. For example, the type $\exists\alpha, \beta[\alpha \leq \beta].int^\alpha \times (int^\beta \rightarrow int^\psi)$ describes a pair in which the first element flows to the domain of the second element. In our system (as in Mossin's system), universally-quantified types obey a Hindley-Milner discipline, and hence they only appear in type environments. In contrast, existential types are first-class so that their values can be stored in variables and passed to functions. Subtyping is permitted between existentials with the same quantification structure (they must have the same α-convertible quantified labels) but differing constraints. While this restriction reduces the power of the system slightly, it aids in our inference algorithm.

We have proven that our polymorphically-constrained type system is sound (Appendix A), but such type systems are often awkward to implement efficiently in practice. Thus the second key contribution of this work is the development of a label flow inference system (Section 4) based on CFL reachability and *instantiation constraints*, in the style of Rehof et al. In this system, subtyping constraints no longer appear on types, but rather are stored "off-to-the-side." Instantiation constraints enable efficient inference of universally- and existentially-quantified types. In our system, functions can be parametric in their arguments except for existential packages, which are passed monomorphically. To support inference, programmers provide annotations specifying which labels in a type are existentially quantified, and our algorithm infers needed constraints. It is an open question whether an efficient algorithm exists that does not require programmer assistance. In inference, we further require that existentially bound labels do not interact we free labels. This requirement means that our inference system is slightly weaker than our checking system, but thus far we have found it necessary to produce a correct inference algorithm.

We have proven that our CFL-based inference system is sound (Appendix B) by reducing it to the system based on polymorphically-constrained types. We also present an $O(n^3)$ inference algorithm that should be efficient in practice.

2 Overview

To understand the use of context-free language reachability for label flow, we review past work concerning context-sensitivity via universal polymorphism. Then we sketch how we adapt this scheme to support existential polymorphism as well.

2.1 Universal Polymorphism and Label Flow

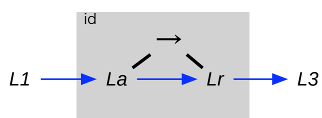
Consider the following program:

$$\begin{array}{l}\texttt{let } id = \lambda a.a \texttt{ in} \\ \quad (id^i\ 1^{L1}) +^{L3} \ldots; \\ \quad (id^j\ 2^{L2}) +^{L4} \ldots\end{array}$$

Here we have annotated the values 1 and 2 with labels $L1$ and $L2$, and we have annotated the + operations with $L3$ and $L4$. Our goal is to determine which labeled values will reach or "flow to" each + operation. We perform a simple monomorphic (context-insensitive) flow analysis as follows. We extend the base type of `id` to a type $int^{La} \rightarrow int^{Lr}$ annotated with labels. Here La is the label on the argument a and Lr is the label on the return. The body of `id` returns its argument, which we express with a constraint $La \leq Lr$. Thus there is "flow" from the label on any integer we pass into `id` to the integer returned by `id`. Whenever we call a function, we generate a constraint to capture the flow of values into and out of the function. Here at the first call to `id` (ignore the superscript on `id` for now), we make two constraints: $L1 \leq La$ and $Lr \leq L3$. Putting these constraints together with the previous one, we get

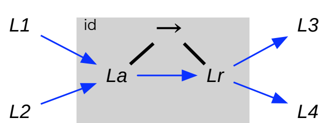
$$L1 \leq La \leq Lr \leq L3$$

and thus by transitivity we know that $L1$ flows to $L3$—which answers the question, "What labeled integers might the first argument of $+^{L3}$ evaluate to?" We can also think of these constraints as edges in a directed graph, where the nodes of the graph are labels and if $L \leq L'$ then there is an edge from L to L'. Thus the above set of constraints is a path through a graph:



Here we have also included edges representing the type structure of *id*. We call such a graph a *flow graph*.[1]

Notice that if we continue this process for the second call to `id`, we get sound but conservative flow. The second call generates the constraints $L2 \leq La$ and $Lr \leq L4$. Putting this together with the previous constraints yields the following graph:

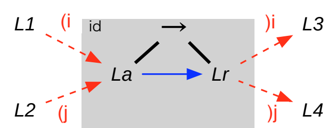


here we see that $L2$ flows to $L4$, which will happen at run time, but the graph also suggests that $L1$ flows to $L4$ and $L2$ flows to $L3$, which cannot actually happen. The analysis has lost precision by conflating the two calls together.

[1] To aid clarity, we draw constraint graphs in color, so viewing the electronic version of this paper may be helpful.

This loss of precision can be corrected by adding *context sensitivity* or *universal polymorphism* to the analysis so that we can differentiate between the flow generated by each call to `id`. Formally, we can give `id` a *polymorphically constrained type* $\forall La, Lr[La \le Lr].int^{La} \to int^{Lr}$, meaning that when this function is called, it produces flow from La to Lr according to its actual arguments. Every time `id` is called in the program body, its type and constraints are *instantiated* to that specific context in which it is called. At the call $\texttt{id}_1$, we instantiate the constraint as $\{La \le Lr\}[La \mapsto L1, Lr \mapsto L3]$, which yields the constraint $L1 \le L3$. At the call $\texttt{id}_2$, we instantiate La to $L2$ and Lr to $L4$, yielding the constraint $L2 \le L4$. Since we made two copies of the constraints for the two different calls, there is no longer any spurious flow. We have effectively inlined the constraints at each call site.

While this technique is effective, it requires explicit constraint copying (to inline them at each call site), which can be difficult to implement, especially if we wish to support polymorphic recursion. An alternative technique is to label edges in the constraint graph and perform CFL reachability to compute flow, as suggested by Rehof et al [8, 26]. In this solution, we label edges at call sites with parentheses indexed by the call. For the example program, we would produce the following flow graph:



In this graph, normal flow is shown using unlabeled edges. Data flow for calls is modeled using edges labeled with indexed parentheses. Here edges from the call id^i are labeled with $(_i$ for inputs and $)_i$ for outputs, and similarly for the call id^j. When we look for paths through the graph, we only accept paths that do not have mismatched parentheses. In this case, the paths from $L1$ to $L3$ and from $L2$ to $L4$ are matched, while the other paths are mismatched and hence not considered. Intuitively, they correspond to unrealizable call-return sequences [14]. The result is a system that supports universal polymorphism.

2.2 Existential Polymorphism and Label Flow

The contribution of this paper is to show how to encode existential quantification into CFL reachability graphs. As a result, we are able to model uses of certain data structures more precisely. Consider the example shown in Figure 1. In this program, we create two functions f and g that add an unspecified value to their argument. We have labeled the $+$ operators in these functions with labels $L3$ and $L4$, respectively. As usual, we wish to determine what integers may flow to these operators, i.e., what integers are passed in as arguments a and b. In the third line of this program we create existentially-quantified pairs using `pack` operations in which f is paired with 1 (labeled $L1$) and g with 2 (labeled $L2$). Using an `if`, we then conflate these two pairs by binding to p. In the last line we use p by applying its first component to its second component.[2]

Notice that in this case, no matter which pair p is assigned, the function f is only ever applied to 1, and the function g is only ever applied to 2. Figure 1 also shows the constraint graph generated for this example program. Similarly to universal types, we model existential types using edges labeled with subscripted parentheses. In this case, when we pack the pair $(f, 1)$, instead of normal

[2]Our examples use pattern matching combined with `unpack` for simplicity of presentation, but the formal language in Section 3 uses projection and `unpack` operators.

let $f = \lambda a.a +^{L3} \cdots$ in
let $g = \lambda b.b +^{L4} \cdots$ in
let $p =$ if0 $\cdots$ then
  pack^i $(f, 1^{L1})$
 else
  pack^j $(g, 2^{L2})$ in
unpack $(p1, p2) = p$ in
  $p1$ @ $p2$

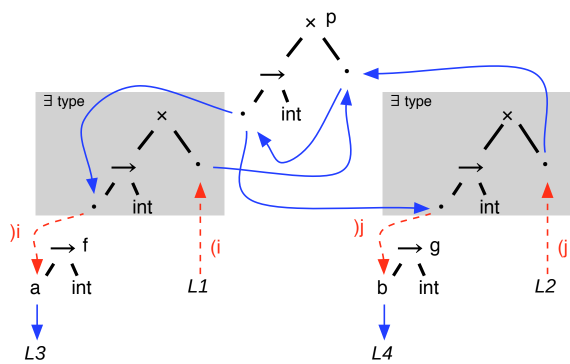


Figure 1: Example Program and Flow Graph

flow edges we generate edges labeled by i-parentheses, and similarly we generate edges labeled with j-parentheses when we pack the pair $(g, 2)$.

To compute the flow for this graph, we again propagate labels along paths with no mismatched parentheses. For example, in this graph there is a path

$$L2 \rightarrow^{(j} \bullet \rightarrow \cdots \rightarrow \bullet \rightarrow^{)j} b \rightarrow L4$$

from which we can conclude that $L2$, i.e., the value 2, may flow to $L4$, i.e., the addition operation in g. There is similarly a path from $L1$ to $L3$.

However, notice crucially that there is no path from $L2$ to $L3$. This corresponds to our observation that f is never applied to 2. Similarly there is no path from $L1$ to $L4$. In a label flow system without existentials, the edges labeled with parentheses would have been unlabeled, resulting in spurious flow. Thus in this case, existential quantification permits conflating two existential packages without losing precision about the use of their members.

Intuitively, for universal quantification, we were able to generalize the type of *id* because *id* is polymorphic in the label it is called with—whatever it is called with, it returns. For this example with existential polymorphism, we could generalize the type of the pairs because the *rest of the program* is polymorphic in the pairs—whichever pair is used by the program, the first element is always applied to the second. It is this duality that allows us to encode both universal and existential polymorphism using the same technique.

3 Label Flow with Polymorphically Constrained Types

We begin studying label flow in the context of a traditional polymorphically-constrained type system Copy, which is Mossin's label flow system extended to model existential types. Note that our system supports label polymorphism but not polymorphism in the type structure.

Figure 2 shows the language used throughout the paper. In this language, constructors and destructors are annotated with *constant labels* L. (In this system we write function application with @ to provide a position on which to write a label.) The goal of our type system is to determine which constructor labels flow to which destructor labels. For example, in the expression $(\lambda^L x.e)@^{L'}e'$, the label L flows to the label L'. Expressions include binding constructs let and

$$
\begin{array}{rcl}
e & ::= & n^L \mid x \mid \lambda^L x.e \mid e_1 @^L e_2 \mid \texttt{if0}^L\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid (e_1, e_2)^L \mid e.^L j \\
& \mid & \texttt{let}\ f = e_1\ \texttt{in}\ e_2 \mid \texttt{fix}\, f.e_1 \mid f^i \mid \texttt{pack}^{L,i}\ e \mid \texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2 \\
L & ::= & \langle \text{constant labels} \rangle
\end{array}
$$

Figure 2: Language Syntax

$$
\begin{array}{lrcl}
\text{types} & \tau & ::= & int^l \mid \tau \rightarrow^l \tau \mid \tau \times^l \tau \mid \exists^l \vec{\alpha}[C].\tau \\
\text{type schemes} & \sigma & ::= & \forall \vec{\alpha}[C].\tau \\
\text{labels} & l & ::= & L \mid \alpha \\
\text{constraints} & C & ::= & \emptyset \mid \{l \leq l\} \mid C \cup C
\end{array}
$$

Figure 3: COPY type definitions

fix, which introduce universal polymorphism. Each use of a universally quantified function f^i is indexed by an *instantiation site* i. Expressions also include existential packages, which are created with indexed $\texttt{pack}^i$ and consumed with unpack. Instantiation sites are ignored in this section, but are used in Section 4.

Figure 3 shows the types used in the COPY system, which include integers, functions, pairs and existentially quantified types. All types are annotated with flow labels l, which may be either constant labels L from the program text or label variables α. Universally-quantified functions are given polymorphically-constrained types of the form $\forall \vec{\alpha}[C].\tau$. Here C is a set of *flow constraints* of the form $l \leq l'$. In our type rules, we also use *substitutions* ϕ that map label variables to labels. Intuitively, the type $\forall \vec{\alpha}[C].\tau$ stands for any type $\phi(\tau)$ where $\phi(C)$ is satisfied, for any substitution ϕ. When $l \leq l'$, we say that label l "flows to" label l'. The type $\exists^l \vec{\alpha}[C].\tau$ stands for the type $\phi(\tau)$ where constraints $\phi(C)$ are satisfied, for some substitution ϕ. Universal quantification may only appear at the top-level while existential types may appear at any position in a type. We define the free labels of types and environments as usual:

$$
\begin{array}{rcl}
fl(int^l) & ::= & \{l\} \\
fl(\tau_1 \rightarrow^l \tau_2) & ::= & \{l\} \cup fl(\tau_1) \cup fl(\tau_2) \\
fl(\tau_1 \times^l \tau_2) & ::= & \{l\} \cup fl(\tau_1) \cup fl(\tau_2) \\
fl(\exists^l \vec{\alpha}[C].\tau) & ::= & \{l\} \cup ((fl(\tau) \cup fl(C)) \setminus \vec{\alpha}) \\
fl(\Gamma, f : \forall \vec{\alpha}[C].\tau) & ::= & fl(\Gamma) \cup ((fl(\tau) \cup fl(C)) \setminus \vec{\alpha}) \\
fl(C \cup \{l \leq l'\}) & ::= & fl(C) \cup \{l, l'\} \\
fl(\Gamma, x : \tau) & ::= & fl(\Gamma) \cup fl(\tau)
\end{array}
$$

The expression typing rules are presented in Figures 4 and 5. Our system is essentially identical to that of Mossin [7], with new rules for existentials. Judgments have the form $C; \Gamma \vdash_{cp} e : \tau$, meaning in type environment Γ with flow constraints C, expression e has type τ. In these type rules we use $C \vdash l \leq l'$ to mean that the constraint $l \leq l'$ is in the transitive closure of the constraints in C. We write $C \vdash C'$ to mean that constraints in C' are in the transitive closure of C.

$$\text{Id}\ \frac{}{C; \Gamma, x : \tau \vdash_{cp} x : \tau} \qquad \text{Int}\ \frac{C \vdash L \leq l}{C; \Gamma \vdash_{cp} n^L : int^l}$$

$$\text{Lam}\ \frac{C; \Gamma, x : \tau \vdash_{cp} e : \tau' \quad C \vdash L \leq l}{C; \Gamma \vdash_{cp} \lambda^L x.e : \tau \rightarrow^l \tau'} \qquad \text{App}\ \frac{\begin{array}{c} C; \Gamma \vdash_{cp} e_1 : \tau \rightarrow^l \tau' \\ C; \Gamma \vdash_{cp} e_2 : \tau \quad C \vdash l \leq L \end{array}}{C; \Gamma \vdash_{cp} e_1 @^L e_2 : \tau'}$$

$$\text{Pair}\ \frac{\begin{array}{c} C; \Gamma \vdash_{cp} e_1 : \tau_1 \quad C; \Gamma \vdash_{cp} e_2 : \tau_2 \\ C \vdash L \leq l \end{array}}{C; \Gamma \vdash_{cp} (e_1, e_2)^L : \tau_1 \times^l \tau_2} \qquad \text{Proj}\ \frac{\begin{array}{c} C; \Gamma \vdash_{cp} e : \tau_1 \times^l \tau_2 \\ C \vdash l \leq L \quad j = 1, 2 \end{array}}{C; \Gamma \vdash_{cp} e.^L j : \tau_j}$$

$$\text{Cond}\ \frac{\begin{array}{c} C; \Gamma \vdash_{cp} e_0 : int^l \quad C \vdash l \leq L \\ C; \Gamma \vdash_{cp} e_1 : \tau \quad C; \Gamma \vdash_{cp} e_2 : \tau \end{array}}{C; \Gamma \vdash_{cp} \texttt{if0}^L\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau} \qquad \text{Sub}\ \frac{\begin{array}{c} C; \Gamma \vdash_{cp} e : \tau_1 \\ C; \emptyset \vdash \tau_1 \leq \tau_2 \end{array}}{C; \Gamma \vdash_{cp} e : \tau_2}$$

Figure 4: Copy Monomorphic Rules

Figure 4 contains the monomorphic typing rules, which are as in the standard λ calculus except for the addition of labels and subtyping. The constructor rules ([Int], [Lam], and [Pair]) require $C \vdash L \leq l$, i.e., the constructor label L must flow to the corresponding label of the constructed type. The destructor rules ([Cond], [App], and [Proj]) require the converse. The subtyping rule [Sub] is discussed below.

Figure 5 contains the polymorphic typing rules. Universal polymorphism is introduced in [Let] and [Fix]. As is standard, we allow generalization only of label variables that are not free in the type environment Γ. Notice that in both these rules, the constraints C' that we use to type check e_1 become the bound constraints in the polymorphic type. Whenever a variable with a universally quantified type is used in the program text, its type is *instantiated*. The [Inst] rule can only be applied if the instantiation $C'[\vec{\alpha} \mapsto \vec{l}]$ of the polymorphic type's constraints is included in the current flow constraints C at that point.

Existentially quantified types are manipulated using `pack` and `unpack`. Since these existential packages are passed around the program, we label them with concrete labels L as with other constructors and destructors. To understand [Pack] and [Unpack], recall that $\forall$ and $\exists$ are dual notions. Notice that $\forall$ introduction ([Let]) restricts what can be universally quantified, and instantiation occurs at $\forall$ elimination ([Inst]). Thus intuitively, $\exists$ introduction ([Pack]) should perform instantiation, and $\exists$ elimination ([Unpack]) should restrict what can be existentially quantified.

In the rule [Pack], an expression e with a concrete type $\tau[\vec{\alpha} \mapsto \vec{l}]$ is abstracted to a type $\exists^l \vec{\alpha}[C'].\tau$. The constraint system C' bound in the existential type is expressed over the quantified variables $\vec{\alpha}$. Intuitively, these constraints C' are determined by how the existential package is used. [Pack] can only be applied if the current constraint system C entails the abstract constraints C' mapped back to concrete labels $\vec{l}$.

In the rule [Unpack], we bind the contents of the type to x in the scope of e_2. This rule places two restrictions on the existential package. First, we must be able to type check e_2 with the constraints $C' \cup C$. Thus, any constraints among the existentially bound labels $\vec{\alpha}$ that are required

$$\text{[Let]}\ \frac{C'; \Gamma \vdash_{cp} e_1 : \tau_1 \qquad C; \Gamma, f : \forall \vec{\alpha}[C'].\tau_1 \vdash_{cp} e_2 : \tau_2 \qquad \vec{\alpha} \subseteq (\mathit{fl}(\tau_1) \cup \mathit{fl}(C')) \setminus \mathit{fl}(\Gamma)}{C; \Gamma \vdash_{cp} \texttt{let}\ f = e_1\ \texttt{in}\ e_2 : \tau_2}$$

$$\text{[Fix]}\ \frac{C'; \Gamma, f : \forall \vec{\alpha}[C'].\tau \vdash_{cp} e : \tau \qquad C \vdash C'[\vec{\alpha} \mapsto \vec{l}] \qquad \vec{\alpha} \subseteq (\mathit{fl}(\tau) \cup \mathit{fl}(C')) \setminus \mathit{fl}(\Gamma)}{C; \Gamma \vdash_{cp} \texttt{fix}\ f.e : \tau[\vec{\alpha} \mapsto \vec{l}]}$$

$$\text{[Inst]}\ \frac{C \vdash C'[\vec{\alpha} \mapsto \vec{l}]}{C; \Gamma, f : \forall \vec{\alpha}[C'].\tau \vdash_{cp} f^i : \tau[\vec{\alpha} \mapsto \vec{l}]}$$

$$\text{[Pack]}\ \frac{C; \Gamma \vdash_{cp} e : \tau[\vec{\alpha} \mapsto \vec{l}] \qquad C \vdash C'[\vec{\alpha} \mapsto \vec{l}] \qquad C \vdash L \leq l}{C; \Gamma \vdash_{cp} \texttt{pack}^{L,i}\ e : \exists^l \vec{\alpha}[C'].\tau}$$

$$\text{[Unpack]}\ \frac{C; \Gamma \vdash_{cp} e_1 : \exists^l \vec{\alpha}[C'].\tau \qquad C \vdash l \leq L \qquad C \cup C'; \Gamma, x : \tau \vdash_{cp} e_2 : \tau' \qquad \vec{\alpha} \subseteq (\mathit{fl}(\tau) \cup \mathit{fl}(C')) \setminus (\mathit{fl}(\Gamma) \cup \mathit{fl}(C) \cup \mathit{fl}(\tau'))}{C; \Gamma \vdash_{cp} \texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2 : \tau'}$$

Figure 5: COPY Polymorphic Rules

by e_2 must be in C'. Second, we require that the labels $\vec{\alpha}$ do not escape the scope of the `unpack`, which is ensured by the subset constraint in this rule.

We could have chosen slightly different rules for [Let] and [Unpack] and still produced a sound system. In particular, we could have used the rules in Figure 6, in which we allow a union in [Let] but forbid quantification over $\mathit{fl}(C)$, and we disallow a union in [Unpack] but allow quantification over $\mathit{fl}(C)$. Any combination of these possibilities is sound. However, it turns out that the rules in Figure 5 are most useful in proving that our CFL-based inference is sound.

The [Sub] rule in Figure 4 uses the subtyping relation shown in Figure 7. In these judgments, D is a map from labels to integer *depths*, which are initially empty by [Sub]. This mapping is used when subtyping existentials. Aside from these, [Sub-Int], [Sub-Pair], and [Sub-Fun] are all standard. In the rule [Sub-∃], we allow one existential to subtype another only if they have the same binding shape. In this rule, we update D so that bound variables in $\vec{\alpha}$ are given their previous depth plus one. For example, within the existential $\exists^l\{l_2\}[\emptyset].(int^{l_1}, int^{l_2})$ we have $D(l_1) = 0$ and $D(l_2) = 1$, assuming that their starting depths are zero. In rule [Sub-Label-1] we allow subtyping among labels only if they both have depth 0, i.e., they are not bound in existential types. The rule [Sub-Label-2] applies to bound variables, and it requires that they are bound at the same depth and are identical. Intuitively, a type with flow constraints C_1 can be used in any position expecting the same or fewer flows between labels. Thus the rule [Sub-∃] also requires $C_1 \vdash C_2$, meaning the constraints C_1 entail the constraints C_2, or every constraint in C_2 is implied by C_1.

These restrictions forbid some clearly erroneous subtyping judgments such as

$$C \vdash \exists\emptyset[\emptyset].(int^l, int^l) \leq \exists\{l\}[\emptyset].(int^l, int^l)$$

$$[\text{Let}']\ \frac{C \cup C'; \Gamma \vdash_{cp} e_1 : \tau_1 \qquad C; \Gamma, f : \forall \vec{\alpha}[C'].\tau_1 \vdash_{cp} e_2 : \tau_2 \\ \vec{\alpha} \subseteq (\mathit{fl}(\tau_1) \cup \mathit{fl}(C')) \setminus (\mathit{fl}(\Gamma) \cup \mathit{fl}(C))}{C; \Gamma \vdash_{cp} \texttt{let } f = e_1 \texttt{ in } e_2 : \tau_2}$$

$$[\text{Unpack}']\ \frac{C; \Gamma \vdash_{cp} e_1 : \exists^l \vec{\alpha}[C'].\tau \qquad C \vdash l \leq L \\ C'; \Gamma, x : \tau \vdash_{cp} e_2 : \tau' \\ \vec{\alpha} \subseteq (\mathit{fl}(\tau) \cup \mathit{fl}(C')) \setminus (\mathit{fl}(\Gamma) \cup \mathit{fl}(\tau'))}{C; \Gamma \vdash_{cp} \texttt{unpack}^L\ x = e_1 \texttt{ in } e_2 : \tau'}$$

Figure 6: Alternative Copy Polymorphic Rules

$$[\text{Sub-Label-1}]\ \frac{D(l) = D(l') = 0 \\ C \vdash l \leq l'}{C; D \vdash l \leq l'} \qquad [\text{Sub-Pair}]\ \frac{C; D \vdash l \leq l' \\ C; D \vdash \tau_1 \leq \tau_1' \\ C; D \vdash \tau_2 \leq \tau_2'}{C; D \vdash \tau_1 \times^l \tau_2 \leq \tau_1' \times^{l'} \tau_2'}$$

$$[\text{Sub-Label-2}]\ \frac{D(l) > 0}{C; D \vdash l \leq l} \qquad [\text{Sub-Fun}]\ \frac{C; D \vdash l \leq l' \\ C; D \vdash \tau_1' \leq \tau_1 \\ C; D \vdash \tau_2 \leq \tau_2'}{C; D \vdash \tau_1 \rightarrow^l \tau_2 \leq \tau_1' \rightarrow^{l'} \tau_2'}$$

$$[\text{Sub-Int}]\ \frac{C; D \vdash l \leq l'}{C; D \vdash \mathit{int}^l \leq \mathit{int}^{l'}} \qquad [\text{Sub-}\exists]\ \frac{C_1 \vdash C_2 \\ D' = D[l \mapsto D(l) + 1, \forall l \in \vec{\alpha}] \\ C; D' \vdash \tau_1 \leq \tau_2 \\ C; D \vdash l_1 \leq l_2}{C; D \vdash \exists^{l_1} \vec{\alpha}[C_1].\tau_1 \leq \exists^{l_2} \vec{\alpha}[C_2].\tau_2}$$

Figure 7: Copy Subtyping

This example should not be allowed to check in any context, because it would create a constraint between a bound label and an unbound label. However, these restrictions also forbid some valid existential subtyping, and it is an open question whether they can be relaxed while still maintaining efficient CFL reachability-based inference.

We prove soundness for Copy using a standard subject-reduction proof. We begin by defining a standard operational semantics $e \longrightarrow e'$, shown in Figure 8. We assume that all input programs are well-typed with respect to the standard types; hence they can never go wrong in our semantics. We wish to prove that, for any destructor that consumes a value, the actual constructor label that is consumed appears in the set of labels computed by the analysis. If the program is in normal form this is trivial, because there are no more evaluation steps. Hence we prove this statement below for the case when the program takes a single step.

Definition 1 *Suppose $e \longrightarrow e'$ and in the (single step) reduction, the destructor* (`if0`, @, $.j$, `unpack`) *labeled L' consumes the constructor* (n, λ, $(\cdot,\cdot)$, `pack`, *respectively*)*, labeled L. Then we*

$$
\begin{array}{llcll}
(\beta) & (\lambda^L x.e)@^{L'} e' & \longrightarrow & e[x \mapsto e'] & \\
(\delta\text{-if}) & \texttt{if0}^{L'}\ 0^L\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \longrightarrow & e_1 & \\
 & \texttt{if0}^{L'}\ n^L\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \longrightarrow & e_2 & n \neq 0 \\
(\delta\text{-pair}) & (e_1, e_2)^L .^{L'} j & \longrightarrow & e_j & j \in \{1, 2\} \\
(\delta\text{-let}) & \texttt{let}\ f = e'\ \texttt{in}\ e & \longrightarrow & e[f \mapsto e'] & \\
(\delta\text{-exist}) & \texttt{unpack}^{L'}\ x = (\texttt{pack}^{L,i}\ e)\ \texttt{in}\ e' & \longrightarrow & e'[x \mapsto e] & \\
(\delta\text{-fix}) & \texttt{fix}\, f.e & \longrightarrow & e[f \mapsto \texttt{fix}\, f.e] &
\end{array}
$$

$$
(\text{Context}) \qquad \frac{e_1 \longrightarrow e_2}{\mathbb{E}[e_1] \longrightarrow \mathbb{E}[e_2]}
$$

$$
\begin{array}{rcl}
\mathbb{E} & ::= & [] \mid \lambda^L x.\mathbb{E} \mid \mathbb{E}@^L e \mid e@^L \mathbb{E} \mid \texttt{fix}\, f.\mathbb{E} \\
 & \mid & \texttt{if0}^L\ \mathbb{E}\ \texttt{then}\ e\ \texttt{else}\ e' \mid \texttt{if0}^L\ e\ \texttt{then}\ \mathbb{E}\ \texttt{else}\ e' \\
 & \mid & \texttt{if0}^L\ e\ \texttt{then}\ e'\ \texttt{else}\ \mathbb{E} \mid (\mathbb{E}, e)^L \mid (e, \mathbb{E})^L \mid \mathbb{E}.^L j \\
 & \mid & \texttt{let}\ f = \mathbb{E}\ \texttt{in}\ e \mid \texttt{let}\ f = e\ \texttt{in}\ \mathbb{E} \\
 & \mid & \texttt{pack}^{L,i}\ \mathbb{E} \mid \texttt{unpack}^L\ x = \mathbb{E}\ \texttt{in}\ e \mid \texttt{unpack}^L\ x = e\ \texttt{in}\ \mathbb{E}
\end{array}
$$

Figure 8: Operational Semantics

write $C \vdash e \longrightarrow e'$ *if* $C \vdash L \leq L'$. *We also write* $C \vdash e \longrightarrow e'$ *if no value is consumed during reduction (i.e., for* `let` *or* `fix`*).*

Theorem 2 (Soundness) *If* $C; \Gamma \vdash_{cp} e : \tau$ *and* $e \longrightarrow^* e'$, *then* $C \vdash e \longrightarrow^* e'$.

The proof is by induction on the derivation of $C; \Gamma \vdash_{cp} e : \tau$, and appears in Appendix A.

4 CFL-Based Label Flow Inference

Next, we present a label flow inference system CFL in the style of Rehof et al [8, 26]. In this system, constraints C no longer appear on quantified types. Rather, they are collected, along with *instantiation constraints* I, into a single inferred *flow graph*, such as the one shown earlier in Figure 1. We answer flow queries "Does any value at program point l_1 flow to program point l_2?", which we write $l_1 \rightsquigarrow l_2$, using CFL reachability queries on the flow graph.

We assume that the input program is well-typed with respect to standard types, and that instantiation indices i are unique per $\texttt{pack}^i$ or f^i. We also implicitly assume that the programmer has decided which labels should be quantified for each existential type. The algorithm then infers the equivalent of the constraints C from Copy as part of the flow graph.

Types in CFL are given by

$$
\begin{array}{llcl}
\text{types} & \tau & ::= & int^l \mid \tau \to^l \tau \mid \tau \times^l \tau \mid \exists^l \vec{\alpha}.\tau \\
\text{type schemes} & \sigma & ::= & (\forall \vec{\alpha}.\tau, \vec{l}) \\
\text{labels} & l & ::= & L \mid \alpha
\end{array}
$$

In contrast to Copy, polymorphic types do not include a constraint set. Universal types, however, include a set $\vec{l}$ of labels that should *not* be quantified in the type. In fact, we could omit the $\vec{\alpha}$

$$[\text{Id}]\ \frac{}{I; C; \Gamma, x : \tau \vdash_{CFL} x : \tau} \qquad [\text{Int}]\ \frac{C \vdash L \leq l}{I; C; \Gamma \vdash_{CFL} n^L : int^l}$$

$$[\text{Lam}]\ \frac{I; C; \Gamma, x : \tau \vdash_{CFL} e : \tau' \quad C \vdash L \leq l}{I; C; \Gamma \vdash_{CFL} \lambda^L x.e : \tau \rightarrow^l \tau'} \qquad [\text{App}]\ \frac{\begin{array}{c} I; C; \Gamma \vdash_{CFL} e_1 : \tau \rightarrow^l \tau' \\ I; C; \Gamma \vdash_{CFL} e_2 : \tau \\ C \vdash l \leq L \end{array}}{I; C; \Gamma \vdash_{CFL} e_1 @^L e_2 : \tau'}$$

$$[\text{Pair}]\ \frac{\begin{array}{c} I; C; \Gamma \vdash_{CFL} e_1 : \tau_1 \quad I; C; \Gamma \vdash_{CFL} e_2 : \tau_2 \\ C \vdash L \leq l \end{array}}{I; C; \Gamma \vdash_{CFL} (e_1, e_2)^L : \tau_1 \times^l \tau_2} \qquad [\text{Proj}]\ \frac{\begin{array}{c} I; C; \Gamma \vdash_{CFL} e : \tau_1 \times^l \tau_2 \\ C \vdash l \leq L \quad j = 1, 2 \end{array}}{I; C; \Gamma \vdash_{CFL} e.^L j : \tau_j}$$

$$[\text{Cond}]\ \frac{\begin{array}{c} I; C; \Gamma \vdash_{CFL} e_0 : int^l \quad C \vdash l \leq L \\ I; C; \Gamma \vdash_{CFL} e_1 : \tau \quad I; C; \Gamma \vdash_{CFL} e_2 : \tau \end{array}}{I; C; \Gamma \vdash_{CFL} \texttt{if0}^L\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau} \qquad [\text{Sub}]\ \frac{\begin{array}{c} I; C; \Gamma \vdash_{CFL} e : \tau_1 \\ C; \emptyset; \emptyset \vdash \tau_1 \leq \tau_2 \end{array}}{I; C; \Gamma \vdash_{CFL} e : \tau_2}$$

Figure 9: CFL Monomorphic Rules

completely, since for inference we can always choose for a type $(\forall \vec{\alpha}.\tau, \vec{l})$ that $\vec{\alpha} = fl(\tau) \setminus \vec{l}$, but we leave in $\vec{\alpha}$ to make proofs slightly easier. For existential types, we do not include a set $\vec{l}$, because we assume that the programmer has specified which labels are existentially quantified. We will check to see that the specification is correct when existentials are unpacked.

We write our inference rules in a style similar to checking rules but interpret them differently. A constraint $C \vdash l \leq l'$ means that $l \leq l'$ should be added to C. In addition to subtyping constraints, our inference system generates a set I of instantiation constraints of the form $l \preceq_p^i l'$ [8]. This constraint indicates that l has been renamed to l' at instantiation site i; these are introduced by `pack`s and universal type instantiations and are the source of context-sensitivity in label flow. The p indicates a *polarity*, which relates to the flow of data. In particular, when p is $+$ then l flows to l'; when p is $-$ then l' flows to l. Therefore, in our examples (Figure 1 and Figure 13), we draw a constraint $l \preceq_+^i l'$ as an edge $l \xrightarrow{)i} l'$, and we draw a constraint $l \preceq_-^i l'$ as an edge $l' \xrightarrow{(i} l$. In our rules we write $I \vdash l \preceq_p^i l'$ to mean the constraint $l \preceq_p^i l'$ should be added to I.

Judgments in CFL have the form $I; C; \Gamma \vdash e : \tau$. The base monomorphic rules for our system are presented in Figure 9. In these rules, free occurrences of l should be treated as generation of a fresh label variable. For example, in [Int], we choose l to be a fresh variable α. In [Lam], we pick a type τ with fresh label variables in every position. Otherwise, these rules are similar to Figure 4.

Figure 10 presents our polymorphic inference rules. We define $fl(\tau)$ to be the free labels of a type as usual, except $fl((\forall \vec{\alpha}.\tau, \vec{l}) = (fl(\tau) \setminus \vec{\alpha}) \cup \vec{l}$:

$$\begin{array}{rcl} fl(int^l) & = & \{l\} \\ fl(\tau_1 \rightarrow^l \tau_2) & = & fl(\tau_1) \cup fl(\tau_2) \cup \{l\} \\ fl(\tau_1 \times^l \tau_2) & = & fl(\tau_1) \cup fl(\tau_2) \cup \{l\} \\ fl(\exists^l \vec{\alpha}.\tau) & = & (fl(\tau) \setminus \vec{\alpha}) \cup \{l\} \\ fl(\Gamma, f : (\forall \vec{\alpha}.\tau, \vec{l})) & = & fl(\Gamma) \cup (fl(\tau) \setminus \vec{\alpha}) \cup \vec{l} \\ fl(\Gamma, x : \tau) & = & fl(\Gamma) \cup fl(\tau) \end{array}$$

$$
\text{[Let]}\ \frac{\begin{array}{cc} I; C; \Gamma \vdash_{CFL} e_1 : \tau_1 & I; C; \Gamma, f : (\forall\vec{\alpha}.\tau_1, \vec{l}) \vdash_{CFL} e_2 : \tau_2 \\ \vec{\alpha} = \mathit{fl}(\tau_1) \setminus \vec{l} & \vec{l} = \mathit{fl}(\Gamma) \end{array}}{I; C; \Gamma \vdash_{CFL} \mathtt{let}\ f = e_1\ \mathtt{in}\ e_2 : \tau_2}
$$

$$
\text{[Fix]}\ \frac{\begin{array}{c} I; C; \Gamma, f : (\forall\vec{\alpha}.\tau, \vec{l}) \vdash_{CFL} e : \tau \quad \vec{\alpha} = \mathit{fl}(\tau) \setminus \mathit{fl}(\Gamma) \quad \vec{l} = \mathit{fl}(\Gamma) \\ I; \emptyset; \emptyset \vdash \tau \preceq^i_+ \tau' : \phi \quad dom(\phi) = \vec{\alpha} \quad I \vdash \vec{l} \preceq^i_+ \vec{l} \quad I \vdash \vec{l} \preceq^i_- \vec{l} \end{array}}{I; C; \Gamma \vdash_{CFL} \mathtt{fix}\, f.e : \tau'}
$$

$$
\text{[Inst]}\ \frac{\begin{array}{c} I; \emptyset; \emptyset \vdash \tau \preceq^i_+ \tau' : \phi \\ dom(\phi) = \vec{\alpha} \quad I \vdash \vec{l} \preceq^i_+ \vec{l} \quad I \vdash \vec{l} \preceq^i_- \vec{l} \end{array}}{I; C; \Gamma, f : (\forall\vec{\alpha}.\tau, \vec{l}) \vdash_{CFL} f^i : \tau'}
$$

$$
\text{[Pack]}\ \frac{\begin{array}{cc} I; C; \Gamma \vdash_{CFL} e : \tau' & I; \emptyset; \emptyset \vdash \tau \preceq^i_- \tau' : \phi \\ dom(\phi) = \vec{\alpha} & C \vdash L \leq l \end{array}}{I; C; \Gamma \vdash_{CFL} \mathtt{pack}^{L,i}\ e : \exists^l \vec{\alpha}.\tau}
$$

$$
\text{[Unpack]}\ \frac{\begin{array}{c} I; C; \Gamma \vdash_{CFL} e_1 : \exists^l \vec{\alpha}.\tau \qquad I; C; \Gamma, x : \tau \vdash_{CFL} e_2 : \tau' \\ \vec{l} = \mathit{fl}(\Gamma) \cup \mathit{fl}(\exists^l \vec{\alpha}.\tau) \cup \mathit{fl}(\tau') \cup L \quad \vec{\alpha} \subseteq \mathit{fl}(\tau) \setminus \vec{l} \quad C \vdash l \leq L \\ \forall l \in \vec{\alpha}, l' \in \vec{l}.(I; C \not\vdash l \rightsquigarrow_m l' \text{ and } I; C \not\vdash l' \rightsquigarrow_m l) \end{array}}{I; C; \Gamma \vdash_{CFL} \mathtt{unpack}^L\ x = e_1\ \mathtt{in}\ e_2 : \tau'}
$$

Figure 10: CFL Polymorphic Rules

In rules [Let] and [Fix], we bind a variable f to a universally quantified type. As is standard we cannot quantify label variables that are free in the environment Γ, and we represent this fact by setting $\vec{l} = \mathit{fl}(\Gamma)$ in the type $(\forall\vec{\alpha}.\tau_1, \vec{l})$. The [Inst] rule *instantiates* the type of f by creating a copy τ' of τ with fresh label variables and generating an instantiation constraint $\tau \preceq^i_+ \tau'$. Intuitively, the instantiation constraint defines a renaming ϕ such that $\phi(\tau) = \tau'$. All non-quantifiable labels, i.e., all labels in $\vec{l}$, should not be instantiated, which we model by requiring that any such label instantiate to itself.

In [Pack], an existential type is constructed by abstracting a concrete type τ' to abstract type τ. In Copy's [Pack], there is a substitution such that $\tau' = \tau[\vec{\alpha} \mapsto \vec{l}]$. In CFL's [Pack], we express this with an instantiation constraint $I; \emptyset; \emptyset \vdash \tau \preceq^i_- \tau'$, similarly to [Inst]. Notice that the direction of the renaming here is opposite the direction of flow: The labels in τ' flow to the labels of τ, but τ is instantiated to τ'. Hence the instantiation has negative polarity. As another intuition, packing a type into an existential corresponds to passing an argument to "the rest of the program," as if that was universally quantified, which implies negative instantiation.

We implicitly assume that the programmer has picked the set $\vec{\alpha}$ in this rule. In contrast to [Inst], we do not generate any self-loops in [Pack], because we enforce a stronger restriction for escaping variables in [Unpack].

Finally, in Copy's [Pack] rule, we also require that the packed type satisfies $C \vdash C'[\vec{\alpha} \mapsto \vec{l}]$, where C' are the constraints induced by uses of the existential package (i.e., within the scope of

$$\text{[Sub-Index-1]}\;\frac{C \vdash l \leq l'}{C; \emptyset; \emptyset \vdash l \leq l'}$$

$$\text{[Sub-Index-2]}\;\frac{C \vdash l_j \leq l'_j}{C; D_1 \oplus \{l_1, \ldots, l_n\}; D_2 \oplus \{l'_1, \ldots, l'_n\} \vdash l_j \leq l'_j}$$

$$\text{[Sub-Index-3]}\;\frac{C; D_1; D_2 \vdash l \leq l' \qquad l \neq l_i \qquad l' \neq l'_j \qquad \forall i, j \in [1..n]}{C; D_1 \oplus \{l_1, \ldots, l_n\}; D_2 \oplus \{l'_1, \ldots, l'_n\} \vdash l \leq l'}$$

$$\text{[Sub-Int]}\;\frac{C; D_1; D_2 \vdash l \leq l'}{C; D_1; D_2 \vdash int^l \leq int^{l'}}$$

$$\text{[Sub-Pair]}\;\frac{\begin{array}{c} C; D_1; D_2 \vdash l \leq l' \\ C; D_1; D_2 \vdash \tau_1 \leq \tau'_1 \\ C; D_1; D_2 \vdash \tau_2 \leq \tau'_2 \end{array}}{C; D_1; D_2 \vdash \tau_1 \times^l \tau_2 \leq \tau'_1 \times^{l'} \tau'_2}$$

$$\text{[Sub-Fun]}\;\frac{\begin{array}{c} C; D_1; D_2 \vdash l \leq l' \\ C; D_1; D_2 \vdash \tau'_1 \leq \tau_1 \\ C; D_1; D_2 \vdash \tau_2 \leq \tau'_2 \end{array}}{C; D_1; D_2 \vdash \tau_1 \rightarrow^l \tau_2 \leq \tau'_1 \rightarrow^{l'} \tau'_2}$$

$$\text{[Sub-}\exists\text{]}\;\frac{\begin{array}{c} D'_1 = D_1 \oplus \vec{\alpha_1} \qquad D'_2 = D_2 \oplus \vec{\alpha_2} \qquad \phi(\vec{\alpha_2}) = \vec{\alpha_1} \\ C; D'_1; D'_2 \vdash \tau_1 \leq \tau_2 \qquad\quad C; D_1; D_2 \vdash l_1 \leq l_2 \end{array}}{C; D_1; D_2 \vdash \exists^{l_1} \vec{\alpha_1}.\tau_1 \leq \exists^{l_2} \vec{\alpha_2}.\tau_2}$$

Figure 11: CFL Subtyping

the `unpacks` it flows to). Here we need not generate such a constraint explicitly, just as we did not generate such constraints in [Inst].

In [Unpack], the abstract existential type is treated as a concrete type for the scope of the unpack. As usual for existential types, no abstract label may escape, or it will break the abstraction. The last hypothesis of [Unpack] enforces a strong restriction so that not only may abstract labels not escape, but they may not constrain any escaping labels in any way. We express this restriction by requiring that there are no matched flows (see below) between any labels in $\vec{\alpha}$ and any labels in $\vec{l}$, which is the set of labels that escape. This includes free labels of the environment Γ, free labels of the existential type that is unpacked $\exists^l \vec{\alpha}.\tau$, free labels of the result type τ', and constants L.

The subtyping relation used in [Sub] is defined in Figure 11. Subtyping judgments have the form $C; D_1; D_2 \vdash \tau_1 \leq \tau_2$. In these rules, D_1 and D_2 are sequences that are used to ensure that labels in corresponding positions in τ_1 and τ_2 are existentially quantified identically, in the same sense as Figure 7. In [Sub-$\exists$], each time we recurse under an existential we add $\vec{\alpha_i}$ (the variables bound in the existential) to D_i. In [Sub-Index-1], we allow constraints between arbitrary labels when the D_i are empty. However, in [Sub-Index-2], we allow subtyping among labels that appear

$$[\text{Inst-Index-1}]\ \frac{I \vdash l \preceq_p^i l' \qquad \{(l, l')\} \in \phi}{I; \emptyset \vdash l \preceq_p^i l' : \phi}$$

$$[\text{Inst-Index-2}]\ \frac{}{I; D \oplus \{l_1, \ldots, l_n\} \vdash l_j \preceq_p^i l_j : \emptyset}$$

$$[\text{Inst-Index-3}]\ \frac{I; D \vdash l \leq l' : \phi \qquad l, l' \neq l_i \qquad \forall i \in [1..n]}{I; D \oplus \{l_1, \ldots, l_n\} \vdash l \preceq_p^i l' : \phi}$$

$$[\text{Inst-Int}]\ \frac{I; D \vdash l \preceq_p^i l' : \phi}{I; D \vdash int^l \preceq_p^i int^{l'} : \phi}$$

$$[\text{Inst-Pair}]\ \frac{\begin{array}{c} I; D \vdash l \preceq_p^i l' : \phi \\ I; D \vdash \tau_1 \preceq_p^i \tau_1' : \phi \qquad I; D \vdash \tau_2 \preceq_p^i \tau_2' : \phi \end{array}}{I; D \vdash \tau_1 \times^l \tau_2 \preceq_p^i \tau_1' \times^{l'} \tau_2' : \phi}$$

$$[\text{Inst-Fun}]\ \frac{\begin{array}{c} I; D \vdash l \preceq_p^i l' : \phi \\ I; D \vdash \tau_1 \preceq_{\bar{p}}^i \tau_1' : \phi \qquad I; D \vdash \tau_2 \preceq_p^i \tau_2' : \phi \end{array}}{I; D \vdash \tau_1 \rightarrow^l \tau_2 \preceq_p^i \tau_1' \rightarrow^{l'} \tau_2' : \phi}$$

$$[\text{Inst-}\exists]\ \frac{D' = D \oplus \vec{\alpha} \qquad I; D' \vdash \tau_1 \preceq_p^i \tau_2 : \phi \qquad I; D \vdash l_1 \preceq_p^i l_2 : \phi}{I; D \vdash \exists^{l_1} \vec{\alpha}.\tau_1 \preceq_p^i \exists^{l_2} \vec{\alpha}.\tau_2 : \phi}$$

Figure 12: CFL Instantiation

in the D_i only if they appear in the exactly the same positions. (Notice that we treat $\vec{\alpha}$ as an ordered sequence.) Intuitively, this subtyping in [Sub-Index-2] corresponds to alpha-renaming of existentially bound variables. The remainder of the rules are standard.

Figure 12 defines the instantiation of an abstract type to a concrete type. Judgments have the form $I; D \vdash \tau \preceq_p^i \tau'$. Intuitively, the D has the same form as the D_i in Figure 11, but we only require one such list of vectors because we do not permit alpha renaming at instantiations. Alpha-renaming of existentials can always be accomplished by applying [Sub] before or after an instantiation.

The rules for integers, pairs, and functions are standard [8]. The polarity p is used to track the direction of data flow across the renaming of τ to τ'. Function arguments are treated contravariantly; the notation $\bar{p}$ flips the polarity of p in [Inst-Fun]. Notice that we flip p but not the instantiation direction, which follows because intuitively we are constructing a renaming from the left-hand side of the instantiation constraint to the right-hand side, and we do not flip the direction of renaming under a function arrow. [Inst-$\exists$] constructs D as [Sub-$\exists$] in Figure 11.

Notice that rule [Inst-Index-2] prevents existentially bound labels from being instantiated. This restriction enables us to work with the standard CFL reachability grammar, but it does mean that existentials are always treated non-context sensitively in function calls. Figure 13 shows one

let $id = \lambda a.a$ in
let $p = \texttt{pack}^i\ (1^{L1}, 2^{L2})$ in
let $z =$
  $(\texttt{unpack}\ (x, y) = p$ in $y)$ in
let $q = \texttt{pack}^j\ (3^{L3}, 4^{L4})$ in
let $w = id^k\ @\ p$ in
  $id^l\ @\ q$

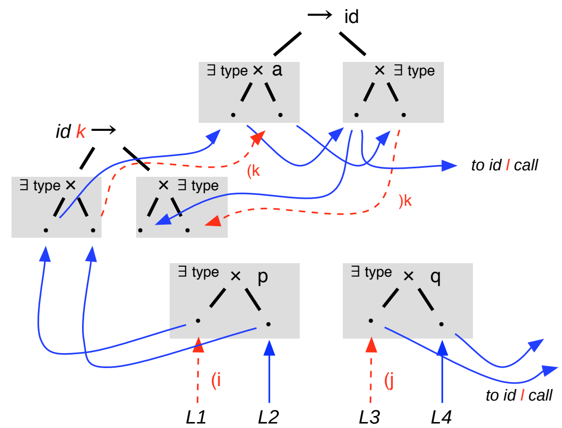


Figure 13: Comparison of Universal and Existential Quantification

$$[\text{Level}]\frac{C \vdash l_1 \leq l_2}{I; C \vdash l_1 \rightsquigarrow_m l_2} \qquad [\text{Trans}]\frac{I; C \vdash l_0 \rightsquigarrow_m l_1 \quad I; C \vdash l_1 \rightsquigarrow_m l_2}{I; C \vdash l_0 \rightsquigarrow_m l_2}$$

$$[\text{Constant}]\frac{}{I; C \vdash L \preceq_p^i L} \qquad [\text{Match}]\frac{I \vdash l_1 \preceq_-^i l_0 \quad I; C \vdash l_1 \rightsquigarrow_m l_2 \quad I \vdash l_2 \preceq_+^i l_3}{I; C \vdash l_1 \rightsquigarrow_m l_3}$$

Figure 14: Flow

example of this. In this program we `pack` two pairs. We can see that it is safe to quantify only the first element of both pairs, because the second element of p escapes the `unpack`. Thus the first element only is instantiated at the `pack`. However, notice that because the second element is not existentially quantified, it can be universally quantified when passed to the function.

Once we have generated constraints, we apply the rules in Figure 14 to close the constraint graph. Here the m subscript indicates matched paths. (Rehof et al also include additional paths that contain unmatched but no mismatched edges. However, here we concern ourselves only with constants, which always produce matched paths.) Rule [Level] states that constraints in C correspond to flow (these are represented as unlabeled edges in the graph). Rule [Trans] adds transitive closure. Rule [Constant] adds a self-loop for every constant label in the program; intuitively we generate these edges because constants are global names. Finally, rule [Match] corresponds to matched paths

$$l_0 \xrightarrow{(_i} l_1 \rightarrow l_2 \xrightarrow{)_i} l_3$$

Using these rules, the flow of constructors to destructors is defined as L flows to L' if $I; C \vdash L \rightsquigarrow_m L'$. After we apply these rules, we also need to check the reachability condition in [Unpack]

The complexity of applying the CFL rules is relative to the size of the derivation. Let n be the size of the type-annotated program. Then aside from the extra side conditions in [Unpack] and the D's in [Sub] and [Inst], the result of Rehof et al shows that applying the rules and computing all flows takes time $O(n^3)$ [8]. To implement [Sub-Index-i] efficiently, rather than maintaining D

sets explicitly and repeatedly traversing them, we temporarily mark each variable with a pair (i, j) indicating its position in D and its position in $\vec{\alpha}$ as we traverse an existential type. We can assume without loss of generality that $|\vec{\alpha}| \leq |fl(\tau)|$ in an existential type, so traversing $\vec{\alpha}$ does not increase the complexity. Then we can select among [Sub-Index-1] and [Sub-Index-2] in constant time for each constraint $C; D_1; D_2 \vdash l \leq l'$, so this does not affect the running time, and similarly for [Inst-Index-i].

Once we have computed all flows, we can easily check the side condition of [Unpack] by walking through the labels in $\vec{\alpha}$ and checking for paths to $\vec{l}$ and vice-versa. Since each set is of size $O(n)$, this takes $O(n^2)$ time, and since there are $O(n)$ uses of [Unpack], in total this takes $O(n^3)$ time. Thus the algorithm as a whole is $O(n^3) + O(n^3) = O(n^3)$.

We have proven that programs that check under CFL are reducible to Copy in Appendix B.

Theorem 3 (Reduction from CFL to Copy) *Let $\mathcal{D}$ be a normal CFL derivation of $I; C; \Gamma \vdash_{CFL} e : \tau$. Let $C^I = \{l_1 \leq l_2 \mid I; C \vdash l_1 \rightsquigarrow_m l_2\}$. Then given a particular translation function $\Psi_{C,I}$, it is the case that $C^I; \Psi_{C,I}(\Gamma) \vdash_{cp} e : \Psi_{C,I}(\tau)$.*

Proof: Theorem 37 shows the reduction under a subset $C^I_{fl(\Gamma)\cup fl(\tau)}$ of C^I, and then Lemma 6 shows that we can prove the same statement under C^I.

□

5 Related Work

As stated in the introduction, our work builds directly on the CFL reachability-based label flow system of Rehof et al [8]. Their cubic-time algorithm for polymorphic recursive label flow inference improves on the previously best-known $O(n^8)$ algorithm [7]. The idea of using CFL reachability in static analysis is due to Reps et al [14], who applied it to lower-order data flow analysis problems.

Existential types can be encoded in System F [16] (p. 377), in which polymorphism is first class. Wells showed that type inference for System F is undecidable [17]. There have been several proposals to support first-class polymorphic type inference using type annotations to avoid the undecidability problem. In ML^{F} [18], programmers annotate function arguments that have universal types. Laufer and Odersky [19] propose an extension to ML with first-class existential types, and Remy [20] similarly proposes an extension with first-class universal types. In both systems, the programmer must explicitly list which type variables to quantify. Existential packs and unpacks correspond to data structure construction and pattern matching, and hence are determined by the program text. In our system, we also require the programmer to specify packs and unpacks as well as which variables are quantified, but in contrast to these three systems we support subtyping (and therefore we need polymorphically constrained types), rather than unification. Note that our solution is restricted to label flow, and only existential types are first-class (but adding first-class universals with programmer-specified quantification would be straightforward). We believe full first-class polymorphic type inference for label flow is decidable, although we do not have a proof.

Simonet [15] extends HM(X) [21], a generic constraint-based type inference framework, to include first-class existential and universal polymorphism with subtyping. Simonet requires the programmer to specify the polymorphically constrained type, including the subtyping constraints C, whereas we infer these (note that we assume we have the whole program). Another key difference is that we use CFL reachability for inference. Once again, however, our system is concerned only with label flow.

In ours and the above systems, both quantification and `pack` and `unpack` must be specified manually. Ideally, an inference algorithm requires no work from the programmer. For example, we could envision a system in which all pairs and their uses are considered as candidate existential types, and the algorithm chooses to quantify only those labels that lead to a minimal flow in the graph. It is an open problem for our system whether such an algorithm exists that is efficient.

6 Conclusion

We have presented two systems for label flow inference that support Hindley-Milner style polymorphism with polymorphic recursion and first-class existential quantification. The system Copy models label flow using polymorphically constrained types in the style of Mossin [7], while our inference system CFL is based on CFL reachability, in the style of Rehof et al [8]. Programmers specify where existentials are introduced and eliminated, and our inference algorithm automatically infers the bounds on their flow, which improves on past work. Our aim is to set a firm theoretical foundation on which to build efficient program analyses that benefit from existential quantification. We believe existential quantification is crucial for supporting efficient modeling of dynamic data structures.

References

[1] Palsberg, J.: Type-based analysis and applications. In: Proceedings of the ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, Snowbird, Utah (2001) 20–27

[2] Fähndrich, M., Rehof, J., Das, M.: Scalable Context-Sensitive Flow Analysis using Instantiation Constraints. [23] 253–263

[3] Das, M., Liblit, B., Fähndrich, M., Rehof, J.: Estimating the Impact of Scalable Pointer Analysis on Optimization. In Cousot, P., ed.: Static Analysis, Eighth International Symposium. Volume 2126 of Lecture Notes in Computer Science., Paris, France, Springer-Verlag (2001) 260–278

[4] Myers, A.C.: Practical Mostly-Static Information Flow Control. [24] 228–241

[5] Kodumal, J., Aiken, A.: The Set Constraint/CFL Reachability Connection in Practice. In: Proceedings of the 2004 ACM SIGPLAN Conference on Programming Language Design and Implementation, Washington, DC (2004) 207–218

[6] Johnson, R., Wagner, D.: Finding User/Kernel Bugs With Type Inference. In: Proceedings of the 13th Usenix Security Symposium, San Diego, CA (2004)

[7] Mossin, C.: Flow Analysis of Typed Higher-Order Programs. PhD thesis, DIKU, Department of Computer Science, University of Copenhagen (1996)

[8] Rehof, J., Fähndrich, M.: Type-Based Flow Analysis: From Polymorphic Subtyping to CFL-Reachability. In: Proceedings of the 28th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, United Kingdom (2001) 54–66

[9] Mitchell, J.C., Plotkin, G.D.: Abstract types have existential type. ACM Transactions on Programming Languages and Systems **10** (1988) 470–502

[10] Das, M.: Unification-based Pointer Analysis with Directional Assignments. [23] 35–46

[11] Xi, H., Pfenning, F.: Dependent Types in Practical Programming. [24] 214–227

[12] Flanagam, C., Abadi, M.: Types for Safe Locking. In Swierstra, D., ed.: 8th European Symposium on Programming. Volume 1576 of Lecture Notes in Computer Science., Amsterdam, The Netherlands, Springer-Verlag (1999) 91–108

[13] Minamide, Y., Morrisett, G., Harper, R.: Typed closure conversion. In: Proceedings of the 23rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, St. Petersburg Beach, Florida (1996) 271–283

[14] Reps, T., Horwitz, S., Sagiv, M.: Precise Interprocedural Dataflow Analysis via Graph Reachability. In: Proceedings of the 22nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California (1995) 49–61

[15] Simonet, V.: An Extension of HM(X) with Bounded Existential and Universal Data Types. [25] 39–50

[16] Pierce, B.C.: Types and Programming Languages. The MIT Press (2002)

[17] Wells, J.B.: Typability and type checking in System F are equivalent and undecidable. Ann. Pure Appl. Logic **98** (1999) 111–156

[18] Botlan, D.L., Rémy, D.: ML^F—Raising ML to the Power of System F. [25] 27–38

[19] Läufer, K., Odersky, M.: Polymorphic type inference and abstract data types. ACM Transactions on Programming Languages and Systems **16** (1994) 1411–1430

[20] Rémy, D.: Programming objects with MLART: An extension to ML with abstract and record types. In: Proceedings of theInternational Symposium on Theoretical Aspects of Computer Science, Sendai, Japan (1994) 321–346

[21] Odersky, M., Sulzmann, M., Wehr, M.: Type inference with constrained types. Theory and Practice of Object Systems **5** (1999) 35–55

[22] Aiken, A., Foster, J.S., Kodumal, J., Terauchi, T.: Checking and Inferring Local Non-Aliasing. In: Proceedings of the 2003 ACM SIGPLAN Conference on Programming Language Design and Implementation, San Diego, California (2003) 129–140

[23] Proceedings of the 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation. In: Proceedings of the 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation, Vancouver B.C., Canada (2000)

[24] Proceedings of the 26th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. In: Proceedings of the 26th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, Texas (1999)

[25] Proceedings of the eighth ACM SIGPLAN International Conference on Functional Programming. In: Proceedings of the eighth ACM SIGPLAN International Conference on Functional Programming, Uppsala, Sweden (2003)

[26] Fähndrich, M., Rehof, J., Das, M.: From Polymorphic Subtyping to CFL Reachability: Context-Sensitive Flow Analysis Using Instantiation Constraints. Technical Report MS-TR-99-84, Microsoft Research (2000)

A Proof of Soundness for Copy

We prove soundness for Copy using a standard subject-reduction style approach. We begin by proving a number of helpful lemmas. First, we need to show that it is sound in several places to weaken a constraint system C to a constraint system C' where $C' \vdash C$. Intuitively this holds because C' contains all the "flows" of C, hence all typing judgments are preserved.

Lemma 4 *If $C' \vdash C$ then $C; D \vdash l \leq l'$ implies $C'; D \vdash l \leq l'$*

Proof: By definition, $C' \vdash C$ requires $C \subseteq C'$. There are two possible ways we could have shown $C; D \vdash l \leq l'$:

Case [Sub-Label-1].

$$\text{[Sub-Label-1]}\frac{D(l) = D(l') = 0 \qquad C \vdash l \leq l'}{C; D \vdash l \leq l'}$$

then $\{l \leq l'\} \subseteq C'$, and we hence we can apply [Sub-Label-1] to show $C'; D \vdash l \leq l'$.

Case [Sub-Label-2].

$$\text{[Sub-Label-2]}\frac{D(l) > 0}{C; D \vdash l \leq l}$$

Obviously, [Sub-Label-2] can be applied for any C, so we also have $C'; D \vdash l \leq l$.

$\square$

Lemma 5 (Constraint weakening in subtyping) *If $C; D \vdash \tau \leq \tau'$ then for any C' such that $C' \vdash C$ it holds that $C'; D \vdash \tau \leq \tau'$*

Proof: By induction on the proof derivation of $C; D \vdash \tau \leq \tau'$.

Case [Sub-Int]. By assumption, we have

$$\text{[Sub-Int]}\frac{C; D \vdash l \leq l'}{C; D \vdash int^l \leq int^{l'}}$$

Then from Lemma 4 we get $C'; D \vdash l \leq l'$ so applying [Sub-Int] again we have $C'; D \vdash int^l \leq int^{l'}$.

Case [Sub-Pair]. By assumption, we have

$$\text{[Sub-Pair]}\frac{\begin{array}{c} C; D \vdash l \leq l' \\ C; D \vdash \tau_1 \leq \tau_1' \\ C; D \vdash \tau_2 \leq \tau_2' \end{array}}{C; D \vdash \tau_1 \times^l \tau_2 \leq \tau_1' \times^{l'} \tau_2'}$$

From Lemma 4 we have $C'; D \vdash l \leq l'$ and by induction we have $C'; D \vdash \tau_1 \leq \tau_1'$ and $C'; D \vdash \tau_2 \leq \tau_2'$. Then applying [Sub-Pair] again, we can show $C'; D \vdash \tau_1 \times^l \tau_2 \leq \tau_1' \times^{l'} \tau_2'$.

Case [Sub-Fun]. Similar to [Sub-Pair].

Case [Sub-∃]. By assumption, we have

$$\text{[Sub-}\exists\text{]}\frac{\begin{array}{c} C_1 \vdash C_2 \\ D' = D[l \mapsto D(l) + 1, \forall l \in \vec{\alpha}] \\ C; D' \vdash \tau_1 \leq \tau_2 \\ C; D \vdash l_1 \leq l_2 \end{array}}{C; D \vdash \exists^{l_1} \vec{\alpha}[C_1].\tau_1 \leq \exists^{l_2} \vec{\alpha}[C_2].\tau_2}$$

From Lemma 4 we have $C'; D \vdash l_1 \leq l_2$. By the induction hypothesis, we have $C'; D \vdash \tau_1 \leq \phi(\tau_2)$ so we can apply [Sub-∃] to prove $C'; D \vdash \exists \vec{\alpha}[C_1].\tau_1 \leq \exists \vec{\alpha}[C_2].\tau_2$

□

Lemma 6 (Constraint weakening in judgment) *If $C; \Gamma \vdash e : \tau$ and $C' \vdash C$ then $C'; \Gamma \vdash e : \tau$*

Proof: By induction on the derivation of $C; \Gamma \vdash e : \tau$. First, observe that if $C \vdash l \leq l'$, then $C' \vdash l \leq l'$, by definition of $C' \vdash C$.

Case [Id], [Int], [App], [Lam], [Pair], [Proj], and [Cond]. The first case is trivial. For the others, apply induction and observe that $C' \vdash L \leq l$ or $C' \vdash l \leq L$, as appropriate.

Case [Sub]. We have

$$\text{[Sub]}\frac{C; \Gamma \vdash_{cp} e : \tau \qquad C; \emptyset \vdash \tau \leq \tau'}{C; \Gamma \vdash_{cp} e : \tau'}$$

Apply induction, and observe that by Lemma 5, $C'; \emptyset \vdash \tau \leq \tau'$.

Case [Let]. We have

$$\text{[Let]}\frac{\begin{array}{c} C''; \Gamma \vdash_{cp} e_1 : \tau_1 \qquad C; \Gamma, f : \forall \vec{\alpha}[C''].\tau_1 \vdash_{cp} e_2 : \tau_2 \\ \vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C'')) \setminus fl(\Gamma) \end{array}}{C; \Gamma \vdash_{cp} \texttt{let}\ f = e_1\ \texttt{in}\ e_2 : \tau_2}$$

By induction, $C'; \Gamma, f : \forall\vec{\alpha}[C''].\tau_1 \vdash_{cp} e_2 : \tau_2$. Thus we can apply [Let] to show $C'; \Gamma \vdash_{cp} \texttt{let}\ f = e_1\ \texttt{in}\ e_2 : \tau_2$.

Case [Fix]. We have

$$[\text{Fix}]\frac{\begin{array}{c}C''; \Gamma, f : \forall\vec{\alpha}[C''].\tau_1 \vdash_{cp} e : \tau_1 \qquad C \vdash C''[\vec{\alpha} \mapsto \vec{l}] \\ \vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C'')) \setminus fl(\Gamma)\end{array}}{C; \Gamma \vdash_{cp} \texttt{fix}\ f.e : \tau_1[\vec{\alpha} \mapsto \vec{l}]}$$

Then since $C' \vdash C$ and $C \vdash C''[\vec{\alpha} \mapsto \vec{l}]$, we have $C' \vdash C''[\vec{\alpha} \mapsto \vec{l}]$. Thus we can apply [Fix] to yield $C'; \Gamma \vdash_{cp} \texttt{fix}\ f.e : \tau_1[\vec{\alpha} \mapsto \vec{l}]$.

Case [Inst]. We have

$$[\text{Inst}]\frac{C \vdash C''[\vec{\alpha} \mapsto \vec{l}]}{C; \Gamma, f : \forall\vec{\alpha}[C''].\tau \vdash_{cp} f^i : \tau[\vec{\alpha} \mapsto \vec{l}]}$$

Then $C' \vdash C''[\vec{\alpha} \mapsto \vec{l}]$, and thus by [Inst], $C'; \Gamma, f : \forall\vec{\alpha}[C''].\tau \vdash_{cp} f^i : \tau[\vec{\alpha} \mapsto \vec{l}]$.

Case [Pack]. We have

$$[\text{Pack}]\frac{\begin{array}{c}C; \Gamma \vdash_{cp} e : \tau[\vec{\alpha} \mapsto \vec{l}] \\ C \vdash C''[\vec{\alpha} \mapsto \vec{l}] \qquad C \vdash L \leq l\end{array}}{C; \Gamma \vdash_{cp} \texttt{pack}^{L,i}\ e : \exists\vec{\alpha}[C''].\tau'}$$

By induction, $C'; \Gamma \vdash_{cp} e : \tau[\vec{\alpha} \mapsto \vec{l}]$. Then $C' \vdash C''[\vec{\alpha} \mapsto \vec{l}]$. As before, we also have $C' \vdash L \leq l$. Thus by [Pack], $C'; \Gamma \vdash_{cp} \texttt{pack}^{L,i}\ e : \exists\vec{\alpha}[C''].\tau'$.

Case [Unpack]. We have

$$[\text{Unpack}]\frac{\begin{array}{c}C; \Gamma \vdash_{cp} e_1 : \exists\vec{\alpha}[C''].\tau_1 \qquad\qquad C \vdash l \leq L \\ C \cup C''; \Gamma, x : \tau_1 \vdash_{cp} e_2 : \tau \\ \vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\tau) \cup fl(C))\end{array}}{C; \Gamma \vdash_{cp} \texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2 : \tau}$$

By induction, $C'; \Gamma \vdash_{cp} e_1 : \exists\vec{\alpha}[C''].\tau_1$. Since $C' \vdash C$, we have $C' \vdash l \leq L$ and $C' \cup C'' \vdash C \cup C''$. Thus also by induction, $C' \cup C''; \Gamma, x : \tau_1 \vdash_{cp} e_2 : \tau$. We can always apply alpha renaming to $\vec{\alpha}$ in C'' and τ_1 so that $fl(C') \cap \vec{\alpha} = \emptyset$, and therefore we have $\vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(C') \cup fl(\tau))$. Thus we can apply [Unpack] to show $C'; \Gamma \vdash_{cp} \texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2 : \tau$.

□

The following lemma is useful for proving soundness of `unpack`. Intuitively, we will use this lemma to reason about subtyping step $C; D \vdash \exists^{l_1}\vec{\alpha}[C_1].\tau_1 \leq \exists^{l_2}\vec{\alpha}[C_2].\tau_2$. Specifically, it will allow us to derive $C; \emptyset \vdash \psi(\tau_1) \leq \psi(\tau_2)$ for a substitution $\psi()$ on $\vec{\alpha}$, because [Sub-Label-2] requires that any labels in $\vec{\alpha}$ have identical occurrences in τ_1 and τ_2,

Lemma 7 *Let $D = D' \circ [l \mapsto 1, \forall l \in \vec{\alpha}])$, where $dom(D') \cap \vec{\alpha} = \emptyset$. Then if $C; D \vdash \tau_1 \leq \tau_2$ and $dom(\psi) = \vec{\alpha}$, then $C; D' \vdash \psi(\tau_1) \leq \psi(\tau_2)$*

Proof: Proof by induction on the derivation $C; D \vdash \tau_1 \leq \tau_2$.

Case [Sub-Label-1]. We have

$$\text{[Sub-Label-1]}\frac{D(l) = D(l') = 0 \qquad C \vdash l \leq l'}{C; D \vdash l \leq l'}$$

But then $l \notin \mathit{dom}(\psi)$ and $l' \notin \mathit{dom}(\psi)$, so $\psi(l) = l$ and $\psi(l') = l'$. But then since $C \vdash l \leq l'$, clearly $C; \emptyset \vdash \psi(l) \leq \psi(l')$.

Case [Sub-Label-2]. We have

$$\text{[Sub-Label-2]}\frac{D(l) > 0}{C; D \vdash l \leq l}$$

This case is trivial, since $C; \emptyset \vdash \psi(l) \leq \psi(l)$.

Case [Sub-Pair]. We have

$$\text{[Sub-Pair]}\frac{\begin{array}{c} C; D \vdash l \leq l' \\ C; D \vdash \tau_1 \leq \tau_1' \\ C; D \vdash \tau_2 \leq \tau_2' \end{array}}{C; D \vdash \tau_1 \times^l \tau_2 \leq \tau_1' \times^{l'} \tau_2'}$$

By induction, $C; \emptyset \vdash \psi(l) \leq \psi(l')$. Also by induction, $C; \emptyset \vdash \psi(\tau_i) \leq \psi(\tau_i')$. Hence by [Sub-Pair], $C; \emptyset \vdash \psi(\tau_1 \times^l \tau_2) \leq \psi(\tau_1' \times^{l'} \tau_2')$.

Case [Sub-Int], [Sub-Fun]. Similar to [Sub-Pair].

Case [Sub-$\exists$]. We have

$$\text{[Sub-}\exists\text{]}\frac{\begin{array}{c} C_1 \vdash C_2 \\ D'' = D[l \mapsto D(l) + 1, \forall l \in \vec{\beta}] \\ C; D'' \vdash \tau_1 \leq \tau_2 \\ C; D \vdash l_1 \leq l_2 \end{array}}{C; D \vdash \exists^{l_1}\vec{\beta}[C_1].\tau_1 \leq \exists^{l_2}\vec{\beta}[C_2].\tau_2}$$

By alpha conversion, we can assume that $\vec{\beta} \cap \mathit{dom}(\psi) = \emptyset$, $\vec{\beta} \cap \mathit{fl}(\mathit{rng}(\psi)) = \emptyset$, and $\vec{\beta} \cap \mathit{dom}(D') = \emptyset$. Notice that $D'' = (D'[l \mapsto D(l) + 1, \forall l \in \vec{\beta}]) \circ [l \mapsto 1, \forall l \in \vec{\alpha}]$. (The order doesn't matter because the domains of the substitutions are all different.) Then by induction, $C; D'[l \mapsto D(l) + 1, \forall l \in \vec{\beta}] \vdash \psi(\tau_1) \leq \psi(\tau_2)$. Further, since $C_1 \vdash C_2$, we have $\psi(C_1) \vdash \psi(C_2)$. Then since we assumed ψ did not replace or capture any variables in $\vec{\beta}$, by [Sub-$\exists$], we have $C; D' \vdash \psi(\exists^{l_1}\vec{\beta}[C_1].\tau_1) \leq \psi(\exists^{l_2}\vec{\beta}[C_2].\tau_2)$.

$\square$

The next lemmas show that in a polymorphically constrained type, we can safely weaken the bound constraints C into C' where $C' \vdash C$. Because existential types are first-class in our system, changing the bound constraints may also change types τ on the right-hand side of a typing judgment.

Let χ range over quantifiers, either $\forall$ or $\exists$. We define *polytypes*(τ) to be the set $\{\chi_i \vec{\alpha_i}[C_i].\tau_i\}$ of occurrences of quantified types in τ. As a shorthand, we write χ_i for the ith element of this set, and we define $\chi_i\langle C'\rangle = \chi_i\vec{\alpha_i}[C' \cup C_i].(\tau_i\langle C'\rangle)$, i.e., we union C' with any bound constraint systems. We implicitly alpha rename bound type variables as necessary to avoid capturing variables in C', i.e., we assume $\vec{\alpha_i} \cap C' = \emptyset$. Here $\tau\langle C\rangle$ is τ where each $\chi_i \in$ *polytypes*(τ) is replaced by $\chi_i\langle C\rangle$. We define *polytypes*(Γ) to be the set of occurrences of quantified types in the range of Γ, and we define $\Gamma\langle C\rangle$ to be Γ with $\langle C\rangle$ applied to the range of Γ.

Lemma 8 *If $C; D \vdash \tau \leq \tau'$, then $C; D \vdash \tau\langle C'\rangle \leq \tau'\langle C'\rangle$.*

Proof: By induction on the derivation of $C \vdash \tau \leq \tau'$. The [Sub-Pair], [Sub-Fun], and [Sub-Int] cases are straightforward.

Case [Sub-$\exists$]. We have

$$\text{[Sub-}\exists\text{]}\frac{\begin{array}{c} C_1 \vdash C_2 \\ D' = D[l \mapsto D(l) + 1, \forall l \in \vec{\alpha_1}] \\ C; D' \vdash \tau_1 \leq \tau_2 \\ C; D \vdash l_1 \leq l_2 \end{array}}{C; D_1; D_2 \vdash \exists^{l_1}\vec{\alpha}[C_1].\tau_1 \leq \exists^{l_2}\vec{\alpha}[C_2].\tau_2}$$

By induction, we have $C; D' \vdash \tau_1\langle C'\rangle \leq \tau_2\langle C'\rangle$. Further, since $C_1 \vdash C_2$, we have $C_1 \cup C' \vdash C_2 \cup C'$. Putting these together and applying [Sub-$\exists$] yields $C; D \vdash \exists^{l_1}\vec{\alpha}[C_1 \cup C'].(\tau_1\langle C'\rangle) \leq \exists^{l_2}\vec{\alpha}[C_2 \cup C'].(\tau_2\langle C'\rangle)$.

$\square$

Lemma 9 (Constraint weakening in polymorphic types) *If $C; \Gamma \vdash e : \tau$, then $C \cup C'; \Gamma\langle C'\rangle \vdash e : \tau\langle C'\rangle$.*

Proof: First, observe that by Lemma 6, we may assume $C \cup C'; \Gamma \vdash e : \tau$. Then the proof proceeds by induction on the derivation of $C \cup C'; \Gamma \vdash e : \tau$.

Case [Id]. We have

$$\text{[Id]}\frac{}{C \cup C'; \Gamma, x : \tau \vdash_{cp} x : \tau}$$

Then trivially we have

$$\text{[Id]}\frac{}{C \cup C'; \Gamma\langle C'\rangle, x : \tau\langle C'\rangle \vdash_{cp} x : \tau\langle C'\rangle}$$

Case [Int]. Trivial.

Case [Lam]. We have

$$\text{[Lam]}\frac{C \cup C'; \Gamma, x : \tau_1 \vdash_{cp} e : \tau_2 \qquad C \cup C' \vdash L \leq l}{C \cup C'; \Gamma \vdash_{cp} \lambda^L x.e : \tau_1 \rightarrow^l \tau_2}$$

By induction we have $C \cup C'; \Gamma\langle C'\rangle, x : \tau_1\langle C'\rangle \vdash_{cp} e : \tau_2\langle C'\rangle$. Then by [Lam], we have $C \cup C'; \Gamma\langle C'\rangle \vdash_{cp} \lambda^L x.e : (\tau_1 \to^l \tau_2)\langle C'\rangle$.

Case [App], [Pair], [Proj], and [Cond]. Similar to [Lam].

Case [Sub]. We have

$$[\text{Sub}]\frac{\begin{array}{c} C \cup C'; \Gamma \vdash_{cp} e : \tau_1 \\ C \cup C'; \emptyset \vdash \tau_1 \leq \tau_2 \end{array}}{C \cup C'; \Gamma \vdash_{cp} e : \tau_2}$$

By induction, $C \cup C'; \Gamma\langle C'\rangle \vdash_{cp} e : \tau_1\langle C'\rangle$. Further, by Lemma 8, we have $C \cup C'; \emptyset \vdash \tau_1\langle C'\rangle \leq \tau_2\langle C'\rangle$. Thus applying [Sub], we have $C \cup C'; \Gamma \vdash_{cp} e : \tau_2\langle C'\rangle$.

Case [Let]. We have

$$[\text{Let}]\frac{\begin{array}{c} C_f; \Gamma \vdash_{cp} e_1 : \tau_1 \qquad C \cup C'; \Gamma, f : \forall\vec{\alpha}[C_f].\tau_1 \vdash_{cp} e_2 : \tau_2 \\ \vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C_f)) \setminus fl(\Gamma) \end{array}}{C \cup C'; \Gamma \vdash_{cp} \texttt{let } f = e_1 \texttt{ in } e_2 : \tau_2}$$

By induction, we have $C_f \cup C'; \Gamma\langle C'\rangle \vdash_{cp} e_1 : \tau_1\langle C'\rangle$. Also by induction, we have $C \cup C'; \Gamma\langle C'\rangle, f : (\forall\vec{\alpha}[C_f].\tau_1)\langle C'\rangle \vdash_{cp} e_2 : \tau_2\langle C'\rangle$ since $C \cup C' \cup C' = C \cup C'$. Since $fl(C') \cap \vec{\alpha} = \emptyset$, we have $(\forall\vec{\alpha}[C_f].\tau_1)\langle C'\rangle = \forall\vec{\alpha}[C_f \cup C'].(\tau_1\langle C'\rangle)$. Further, $fl(\Gamma\langle C'\rangle) = fl(\Gamma) \cup fl(C')$ and $fl(\tau_1\langle C'\rangle) = fl(\tau_1) \cup fl(C')$, hence we have $\vec{\alpha} \subseteq (fl(\tau_1\langle C'\rangle) \cup fl(C_f \cup C')) \setminus fl(\Gamma)\langle C'\rangle$. Thus we can apply [Let] to yield $C \cup C'; \Gamma\langle C'\rangle \vdash_{cp} \texttt{let } f = e_1 \texttt{ in } e_2 : \tau_2\langle C'\rangle$.

Case [Fix]. Similar to [Let].

Case [Inst]. We have

$$[\text{Inst}]\frac{C \cup C' \vdash C_f[\vec{\alpha} \mapsto \vec{l}]}{C \cup C'; \Gamma, f : \forall\vec{\alpha}[C_f].\tau \vdash_{cp} f^i : \tau[\vec{\alpha} \mapsto \vec{l}]}$$

By our alpha-renaming convention, $fl(C') \cap \vec{\alpha} = \emptyset$. Then $(\Gamma, f : \forall\vec{\alpha}[C_f].\tau)\langle C'\rangle = \Gamma\langle C'\rangle, f : \forall\vec{\alpha}[C_f \cup C'].(\tau\langle C'\rangle)$. Clearly $C \cup C' \vdash C_f[\vec{\alpha} \mapsto \vec{l}] \cup C'$, and by our alpha-renaming convention $C_f[\vec{\alpha} \mapsto \vec{l}] \cup C' = (C_f \cup C')[\vec{\alpha} \mapsto \vec{l}]$. Therefore applying [Inst] yields $C \cup C'; (\Gamma, f : \forall\vec{\alpha}[C_f].\tau)\langle C'\rangle \vdash_{cp} f^i : (\tau\langle C'\rangle)[\vec{\alpha} \mapsto \vec{l}]$. Then since by our alpha-renaming convention $(\tau\langle C'\rangle)[\vec{\alpha} \mapsto \vec{l}] = (\tau[\vec{\alpha} \mapsto \vec{l}])\langle C'\rangle$, we have shown the conclusion.

Case [Pack]. We have

$$[\text{Pack}]\frac{C \cup C'; \Gamma \vdash_{cp} e : \tau[\vec{\alpha} \mapsto \vec{l}] \qquad C \cup C' \vdash C_1[\vec{\alpha} \mapsto \vec{l}] \qquad C \cup C' \vdash L \leq l}{C \cup C'; \Gamma \vdash_{cp} \texttt{pack}^{L,i}\ e : \exists^l\vec{\alpha}[C_1].\tau}$$

By induction, $C \cup C'; \Gamma\langle C'\rangle \vdash_{cp} (\tau[\vec{\alpha} \mapsto \vec{l}])\langle C'\rangle$ since $C \cup C' \cup C' = C \cup C'$. By our alpha-renaming convention, $fl(C') \cap \vec{\alpha} = \emptyset$, so $(\tau[\vec{\alpha} \mapsto \vec{l}])\langle C'\rangle = (\tau\langle C'\rangle)[\vec{\alpha} \mapsto \vec{l}]$. Clearly $C \cup C' \vdash C_1[\vec{\alpha} \mapsto \vec{l}] \cup C'$, and also by our alpha-renaming convention $C_1[\vec{\alpha} \mapsto \vec{l}] \cup C' = (C_1 \cup C')[\vec{\alpha} \mapsto \vec{l}]$. Thus applying [Pack] we have $C \cup C'; \Gamma\langle C'\rangle \vdash \texttt{pack}^{L,i}\ e : \exists^l \vec{\alpha}[C_1 \cup C'].(\tau\langle C'\rangle)$. And by our alpha-renaming convention, $\exists^l \vec{\alpha}[C_1 \cup C'].(\tau\langle C'\rangle) = (\exists^l \vec{\alpha}[C_1].\tau)\langle C'\rangle$, so we have shown the conclusion.

Case [Unpack]. We have

$$\text{[Unpack]}\frac{\begin{array}{c} C \cup C'; \Gamma \vdash_{cp} e_1 : \exists^l \vec{\alpha}[C_1].\tau_1 \qquad C \cup C' \vdash l \leq L \\ C \cup C' \cup C_1; \Gamma, x : \tau_1 \vdash_{cp} e_2 : \tau \\ \vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C_1)) \setminus (fl(\Gamma) \cup fl(\tau) \cup fl(C) \cup fl(C')) \end{array}}{C \cup C'; \Gamma \vdash_{cp} \texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2 : \tau}$$

By induction, $C \cup C'; \Gamma\langle C'\rangle \vdash_{cp} e_1 : (\exists^l \vec{\alpha}[C_1].\tau_1)\langle C'\rangle$. By our alpha-renaming convention, $(\exists^l \vec{\alpha}[C_1].\tau_1)\langle C'\rangle = \exists^l \vec{\alpha}[C_1 \cup C'].(\tau_1\langle C'\rangle)$. Also by induction, $C \cup C' \cup C_1; \Gamma\langle C'\rangle, x : \tau_1\langle C'\rangle \vdash_{cp} e_2 : \tau\langle C'\rangle$. Finally, $\vec{\alpha} \subseteq (fl(\tau_1\langle C'\rangle) \cup fl(C_1 \cup C')) \setminus (fl(\Gamma\langle C'\rangle) \cup fl(\tau\langle C'\rangle) \cup fl(C) \cup fl(C'))$ since $fl(\Gamma\langle C'\rangle) = fl(\Gamma) \cup fl(C')$, $fl(\tau_1\langle C'\rangle) = fl(\tau_1) \cup fl(C')$, $fl(\tau\langle C'\rangle) = fl(\tau) \cup fl(C')$, and we assume by alpha-renaming that $fl(C') \cap \vec{\alpha} = \emptyset$. Thus applying [Unpack] yields $C \cup C'; \Gamma \vdash_{cp} \texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2 : \tau\langle C'\rangle$.

□

Next we prove the substitution lemma for monomorphic types. Because in rule [Let], we fixed the set of constraints in the quantified type to be exactly the constraints for e_1 and e_2, respectively, we need a slightly nonstandard lemma: When we replace a variable with an expression, we might need to add the constraints for that expression to quantified types in the environment and in the result type. Hence the definition of $\langle C'\rangle$ above. While we could have used a [Let] rule that is simpler to reason about for soundness, or changed the [Unpack] rule to match [Let], this particular formulation turns out to be very helpful in proving correspondence between the CFL and Copy in Appendix B.

Lemma 10 (Substitution lemma) *If $C; \Gamma, x : \tau' \vdash_{cp} e : \tau$, $C \vdash C'$, and $C'; \Gamma \vdash_{cp} e' : \tau'$, then $C; \Gamma\langle C'\rangle \vdash_{cp} e[x \mapsto e'] : \tau\langle C'\rangle$.*

Proof: The proof proceeds by induction on the derivation of $C; \Gamma, x : \tau' \vdash_{cp} e : \tau$.
Case [Id]. There are two cases. First, if $e = x$, we have

$$\frac{}{C; \Gamma, x : \tau' \vdash_{cp} x : \tau'}$$

Then $\tau = \tau'$, and since $x[x \mapsto e'] = e'$, by our assumption $C'; \Gamma \vdash_{cp} e' : \tau'$ we have $C; \Gamma\langle C'\rangle \vdash_{cp} e' : \tau'\langle C'\rangle$ by Lemmas 6 and 9.

Otherwise, we have

$$\frac{}{C; \Gamma, x : \tau \vdash_{cp} y : \tau}$$

where $y \neq x$. Hence $y \in dom(\Gamma)$, and since $y[x \mapsto e'] = y$, by Lemma 9 we have $C; \Gamma\langle C'\rangle, x : \tau\langle C'\rangle \vdash_{cp} y : \tau\langle C'\rangle$.

Case [Int]. Trivial.

Case [Lam]. We have

$$\text{[Lam]}\frac{C; \Gamma, x : \tau', y : \tau_1 \vdash_{cp} e_2 : \tau_2 \qquad C \vdash L \leq l}{C; \Gamma, x : \tau' \vdash_{cp} \lambda^L y.e_2 : \tau_1 \to^l \tau_2}$$

Using alpha renaming we can assume $y \neq x$, and hence $C; \Gamma, y : \tau_1, x : \tau' \vdash_{cp} e_2 : \tau_2$. Then by induction we have $C; \Gamma\langle C' \rangle, y : \tau_1\langle C' \rangle \vdash_{cp} e_2[x \mapsto e'] : \tau_2\langle C' \rangle$. Thus we can apply [Lam] to yield $C; \Gamma\langle C' \rangle \vdash_{cp} (\lambda^L y.e_2)[x \mapsto e'] : (\tau_1 \to^l \tau_2)\langle C' \rangle$.

Case [App]. We have

$$[\text{App}]\frac{\begin{array}{c} C; \Gamma, x : \tau' \vdash_{cp} e_1 : \tau_2 \to^l \tau \\ C; \Gamma, x : \tau' \vdash_{cp} e_2 : \tau_2 \qquad C \vdash l \leq L \end{array}}{C; \Gamma, x : \tau' \vdash_{cp} e_1 @^L e_2 : \tau}$$

Then by induction, we have $C; \Gamma\langle C' \rangle \vdash_{cp} e_1[x \mapsto e'] : (\tau_2 \to^l \tau)\langle C' \rangle$ and $C; \Gamma\langle C' \rangle \vdash_{cp} e_2[x \mapsto e'] : \tau_2\langle C' \rangle$. Therefore we can apply [App] to yield $C; \Gamma\langle C' \rangle \vdash_{cp} (e_1 @^L e_2)[x \mapsto e'] : \tau\langle C' \rangle$.

Case [Pair], [Proj], [Cond]. Similar to [App].

Case [Sub]. We have

$$[\text{Sub}]\frac{\begin{array}{c} C; \Gamma, x : \tau' \vdash_{cp} e : \tau_1 \\ C; \emptyset \vdash \tau_1 \leq \tau_2 \end{array}}{C; \Gamma, x : \tau' \vdash_{cp} e : \tau_2}$$

By induction, we have $C; \Gamma\langle C' \rangle \vdash_{cp} e[x \mapsto e'] : \tau_1\langle C' \rangle$. By Lemma 5, we have $C; \emptyset \vdash \tau_1\langle C' \rangle \leq \tau_2\langle C' \rangle$. Thus we can apply [Sub] to yield $C; \Gamma\langle C' \rangle \vdash_{cp} e[x \mapsto e'] : \tau_2\langle C' \rangle$.

Case [Let]. We have

$$[\text{Let}]\frac{\begin{array}{c} C''; \Gamma, x : \tau' \vdash_{cp} e_1 : \tau_1 \\ C; \Gamma, x : \tau', f : \forall\vec{\alpha}[C''].\tau_1 \vdash_{cp} e_2 : \tau_2 \\ \vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\tau')) \end{array}}{C; \Gamma, x : \tau' \vdash_{cp} \texttt{let } f = e_1 \texttt{ in } e_2 : \tau_2}$$

By Lemma 6 and induction, we have $C' \cup C''; \Gamma\langle C' \rangle \vdash_{cp} e_1[x \mapsto e'] : \tau_1\langle C' \rangle$. Then since $x \neq f$ (they are in different syntactic categories), by induction we also have $C; \Gamma\langle C' \rangle, f : (\forall\vec{\alpha}[C''].\tau_1)\langle C' \rangle \vdash e_2[x \mapsto e'] : \tau_2\langle C' \rangle$. By our alpha-renaming convention, $fl(C') \cap \vec{\alpha} = \emptyset$, so $(\forall\vec{\alpha}[C''].\tau_1)\langle C' \rangle = \forall\vec{\alpha}[C' \cup C''].\tau_1\langle C' \rangle$. Finally,

$$\begin{array}{rcl} \vec{\alpha} & \subseteq & (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\tau')) \\ & \subseteq & (fl(\tau_1) \cup fl(C'')) \setminus fl(\Gamma) \\ & \subseteq & (fl(\tau_1\langle C' \rangle) \cup fl(C'') \cup fl(C')) \setminus fl(\Gamma\langle C' \rangle) \end{array}$$

where the last step holds since we assume $fl(C') \cap \vec{\alpha} = \emptyset$. Hence we can apply [Let] to yield $C; \Gamma\langle C' \rangle \vdash_{cp} (\texttt{let } f = e_1 \texttt{ in } e_2)[x \mapsto e'] : \tau_2\langle C' \rangle$.

Case [Fix]. Similar to [Let] and [Inst].

Case [Inst]. By Lemma 9 we have $C; \Gamma\langle C' \rangle, x : \tau'\langle C' \rangle, f : (\forall\vec{\alpha}[C'' \cup C'].(\tau))\langle C' \rangle \vdash_{cp} f^i : \tau\langle C' \rangle$. Then since $f_i[x \mapsto e'] = f_i$ (note we assume different syntactic forms for functions and local variables), we trivially have $C; \Gamma\langle C' \rangle, f : (\forall\vec{\alpha}[C'' \cup C'].(\tau))\langle C' \rangle \vdash_{cp} f^i[x \mapsto e'] : \tau\langle C' \rangle$

Case [Pack]. We have

$$[\text{Pack}]\frac{C; \Gamma, x : \tau' \vdash_{cp} e : \tau[\vec{\alpha} \mapsto \vec{l}] \qquad C \vdash C''[\vec{\alpha} \mapsto \vec{l}] \qquad C \vdash L \leq l}{C; \Gamma, x : \tau' \vdash_{cp} \texttt{pack}^{L,i}\ e : \exists^l \vec{\alpha}[C''].\tau}$$

By induction, $C;\Gamma\langle C'\rangle \vdash_{cp} e[x \mapsto e'] : (\tau[\vec{\alpha} \mapsto \vec{l}])\langle C'\rangle$. Since we assume by alpha renaming that $fl(C') \cap \vec{\alpha} = \emptyset$, we have $(\tau[\vec{\alpha} \mapsto \vec{l}])\langle C'\rangle = (\tau\langle C'\rangle)[\vec{\alpha} \mapsto \vec{l}]$. Further, since $C \vdash C'$, we have $C \vdash C' \cup C''[\vec{\alpha} \mapsto \vec{l}]$, and again since $fl(C') \cap \vec{\alpha} = \emptyset$ we have $C \vdash (C' \cup C'')[\vec{\alpha} \mapsto \vec{l}]$. Then applying [Pack] yields $C;\Gamma\langle C'\rangle \vdash_{cp} (\texttt{pack}^{L,i}\ e)[x \mapsto e'] : (\exists^l \vec{\alpha}[C''].\tau)\langle C'\rangle$.

Case [Unpack]. We have

$$\text{[Unpack]}\frac{\begin{array}{c} C;\Gamma, x : \tau' \vdash_{cp} e_1 : \exists^l \vec{\alpha}[C''].\tau_1 \qquad\qquad C \vdash l \leq L \\ C \cup C'';\Gamma, x : \tau', y : \tau_1 \vdash_{cp} e_2 : \tau \\ \vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\tau') \cup fl(\tau) \cup fl(C)) \end{array}}{C;\Gamma, x : \tau' \vdash_{cp} \texttt{unpack}^L\ y = e_1\ \texttt{in}\ e_2 : \tau}$$

Assume by alpha renaming that $x \neq y$. Then by induction, $C;\Gamma\langle C'\rangle \vdash_{cp} e_1[x \mapsto e'] : (\exists^l \vec{\alpha}[C''].\tau_1)\langle C'\rangle$. By our alpha renaming convention, we assume $fl(C') \cap \vec{\alpha} = \emptyset$, hence $(\exists^l \vec{\alpha}[C''].\tau_1)\langle C'\rangle = \exists^l \vec{\alpha}[C' \cup C''].(\tau_1\langle C'\rangle)$. Since $C \vdash C'$, we have $C \cup C' \cup C'' = C \cup C''$. Thus by Lemma 6 and induction we have $C \cup C' \cup C'';\Gamma\langle C'\rangle, y : \tau_1\langle C'\rangle \vdash_{cp} e_2[x \mapsto e'] : \tau\langle C'\rangle$. Finally,

$$\begin{array}{rcl} \vec{\alpha} & \subseteq & (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\tau') \cup fl(\tau) \cup fl(C)) \\ & \subseteq & (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\tau) \cup fl(C)) \\ & \subseteq & (fl(\tau_1\langle C'\rangle) \cup fl(C'') \cup fl(C')) \setminus (fl(\Gamma\langle C'\rangle) \cup fl(\tau\langle C'\rangle) \cup fl(C)) \end{array}$$

again because we assume $fl(C') \cap \vec{\alpha} = \emptyset$. Hence we can apply [Unpack] to yield $C;\Gamma\langle C'\rangle \vdash_{cp} (\texttt{unpack}^L\ y = e_1\ \texttt{in}\ e_2)[x \mapsto e'] : \tau\langle C'\rangle$.

$\square$

Lemma 11 (Polymorphic substitution lemma) *If $C;\Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} e : \tau$ and $C';\Gamma \vdash_{cp} e' : \tau'$ where $\vec{\alpha} \cap fl(\Gamma) = \emptyset$, then $C;\Gamma \vdash_{cp} e[f \mapsto e'] : \tau$.*

Proof: By induction on the derivation of $C;\Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} e : \tau$.

Case [Id]. Trivial, since $x[f \mapsto e'] = x$ (note we assume different syntactic forms for functions and local variables).

Case [Int]. Trivial.

Case [Lam]. We have

$$\text{[Lam]}\frac{C;\Gamma, f : \forall\vec{\alpha}[C'].\tau', x : \tau_1 \vdash_{cp} e : \tau_2 \qquad C \vdash L \leq l}{C;\Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} \lambda^L x.e : \tau_1 \to^l \tau_2}$$

By alpha conversion, we can assume $\vec{\alpha} \cap fl(\tau_1) = \emptyset$ and $C';\Gamma, x : \tau_1 \vdash_{cp} e' : \tau'$. Then since $x \neq f$, by induction we have $C;\Gamma, x : \tau_1 \vdash_{cp} e[f \mapsto e'] : \tau_2$. But then applying [Lam] we have $C;\Gamma \vdash_{cp} (\lambda^L x.e)[f \mapsto e'] : \tau_1 \to^l \tau_2$.

Case [App]. We have

$$\text{[App]}\frac{\begin{array}{c} C;\Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} e_1 : \tau_2 \to^l \tau_1 \\ C;\Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} e_2 : \tau_2 \qquad C \vdash l \leq L \end{array}}{C;\Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} e_1 @^L e_2 : \tau_1}$$

By induction, we have $C; \Gamma \vdash_{cp} e_1[f \mapsto e'] : \tau_2 \to^l \tau_1$ and $C; \Gamma \vdash_{cp} e_2[f \mapsto e'] : \tau_2$. Then applying [App] yields $C; \Gamma \vdash_{cp} (e_1 @^L e_2)[f \mapsto e'] : \tau_1$.

Case [Pair], [Proj], [Cond], [Sub]. Analogous to [App].

Case [Let]. We have

$$[\text{Let}]\frac{\begin{array}{c} C''; \Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} e_1 : \tau_1 \\ C; \Gamma, f : \forall\vec{\alpha}[C'].\tau', g : \forall\vec{\beta}[C''].\tau_1 \vdash_{cp} e_2 : \tau_2 \\ \vec{\beta} \subseteq (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\forall\vec{\alpha}[C'].\tau')) \end{array}}{C; \Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} \texttt{let } g = e_1 \texttt{ in } e_2 : \tau_2}$$

By induction, $C''; \Gamma \vdash_{cp} e_1[f \mapsto e'] : \tau_1$. Assuming by alpha renaming that $f \neq g$, by induction we also have $C; \Gamma, g : \forall\vec{\beta}[C''].\tau_1 \vdash_{cp} e_2[f \mapsto e'] : \tau_2$. Finally,

$$\begin{array}{rcl} \vec{\beta} & \subseteq & (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\forall\vec{\alpha}[C'].\tau')) \\ & \subseteq & (fl(\tau_1) \cup fl(C'')) \setminus fl(\Gamma) \end{array}$$

so we can apply [Let] to show $C; \Gamma \vdash_{cp} (\texttt{let } g = e_1 \texttt{ in } e_2)[f \mapsto e'] : \tau_2$.

Case [Fix]. Similar to [Let] and [Inst].

Case [Inst]. There are two cases. If $e \neq f$, then the conclusion holds trivially, since $e[f \mapsto e'] = e$. Otherwise, we have

$$[\text{Inst}]\frac{C \vdash C'[\vec{\alpha} \mapsto \vec{l}]}{C; \Gamma, f : \forall\vec{\alpha}[C'].\tau \vdash_{cp} f^i : \tau[\vec{\alpha} \mapsto \vec{l}]}$$

By assumption, $C'; \Gamma \vdash_{cp} e' : \tau'$. Then $C'[\vec{\alpha} \mapsto \vec{l}]; \Gamma[\vec{\alpha} \mapsto \vec{l}] \vdash_{cp} e' : \tau[\vec{\alpha} \mapsto \vec{l}]$. But since by assumption $\vec{\alpha} \cap fl(\Gamma) = \emptyset$, we then have $C'[\vec{\alpha} \mapsto \vec{l}]; \Gamma \vdash_{cp} e' : \tau[\vec{\alpha} \mapsto \vec{l}]$. But $C \vdash C'[\vec{\alpha} \mapsto \vec{l}]$, and so by Lemma 6, $C; \Gamma \vdash_{cp} e' : \tau[\vec{\alpha} \mapsto \vec{l}]$, and so we have shown the conclusion, since $f^i[f \mapsto e'] = e'$.

Case [Pack]. We have

$$[\text{Pack}]\frac{C; \Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} e : \tau[\vec{\beta} \mapsto \vec{l}] \qquad C \vdash C''[\vec{\beta} \mapsto \vec{l}] \qquad C \vdash L \leq l}{C; \Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} \texttt{pack}^{L,i}\ e : \exists^l \vec{\beta}[C''].\tau}$$

By induction, we have $C; \Gamma \vdash_{cp} e[f \mapsto e'] : \tau[\vec{\beta} \mapsto \vec{l}]$. But then we can apply [Pack] to show $C; \Gamma \vdash_{cp} (\texttt{pack}^{L,i}\ e)[f \mapsto e'] : \exists^l \vec{\beta}[C''].\tau$.

Case [Unpack]. We have

$$[\text{Unpack}]\frac{\begin{array}{c} C; \Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} e_1 : \exists^l \vec{\beta}[C''].\tau_1 \qquad\qquad C \vdash l \leq L \\ C \cup C''; \Gamma, f : \forall\vec{\alpha}[C'].\tau', x : \tau_1 \vdash_{cp} e_2 : \tau \\ \vec{\beta} \subseteq (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\forall\vec{\alpha}[C'].\tau') \cup fl(\tau) \cup fl(C)) \end{array}}{C; \Gamma, f : \forall\vec{\alpha}[C'].\tau' \vdash_{cp} \texttt{unpack}^L\ x = e_1 \texttt{ in } e_2 : \tau}$$

By induction, we have $C; \Gamma \vdash_{cp} e_1[f \mapsto e'] : \exists^l \vec{\beta}[C''].\tau_1$. Also by induction, assuming that $f \neq x$ (since functions are in a different syntactic category), we have $C \cup C''; \Gamma, x : \tau_1 \vdash_{cp} e_2[f \mapsto e'] : \tau$.

Finally,

$$\begin{array}{rcl}\vec{\beta} & \subseteq & (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\forall\vec{\alpha}[C'].\tau') \cup fl(\tau) \cup fl(C)) \\ & \subseteq & (fl(\tau_1) \cup fl(C'')) \setminus (fl(\Gamma) \cup fl(\tau) \cup fl(C))\end{array}$$

Thus we can apply [Unpack] to show $C; \Gamma \vdash (\texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2)[f \mapsto e'] : \tau$.

□

Finally, we can state and prove our soundness theorem. We assume that the program is well-typed with respect to the standard types. Hence, every program is either in normal form or can take a step. We wish to prove that, for any destructor that consumes a value, the actual constructor label that is consumed appears in the set of labels computed by the analysis. If the program is in normal form this is trivial, because there are no more evaulation steps. Hence we prove this statement below for the case when the program takes a single step.

Definition 12 *Suppose $e \longrightarrow e'$ and in the (single step) reduction, the destructor* (`if0`, @, $.j$, `unpack`) *labeled L' consumes the constructor* (n, λ, $(\cdot, \cdot)$, `pack`) *labeled L. Then we write $C \vdash e \longrightarrow e'$ if $C \vdash L \leq L'$. We also write $C \vdash e \longrightarrow e'$ if no value is consumed during reduction (e.g., for* `let` *or* `fix`*).*

Notice that if $C \vdash e \longrightarrow e'$ and $\mathbb{E}[e] \longrightarrow \mathbb{E}[e']$, then $C \vdash \mathbb{E}[e] \longrightarrow \mathbb{E}[e']$, since reducing inside of a context does not change which destructor consumed which constructor. We will use this fact implicitly in the proof below.

Lemma 13 (Preservation) *If $C; \Gamma \vdash_{cp} e : \tau$ and $e \longrightarrow e'$, then $C; \Gamma\langle C\rangle \vdash_{cp} e' : \tau\langle C\rangle$ and $C \vdash e \longrightarrow e'$.*

Proof: The proof is by induction on the derivation of $C; \Gamma \vdash_{cp} e : \tau$.

Case [Id], [Int]. These cases cannot happen, because we assume $e \longrightarrow e'$.

Case [Lam]. In this case, the tern is $\lambda^L x.e$, and the only possible reduction is $\lambda^L x.e \longrightarrow \lambda^L x.e'$. By assumption, we have

$$\text{[Lam]}\frac{C; \Gamma, x : \tau \vdash_{cp} e : \tau' \qquad C \vdash L \leq l}{C; \Gamma \vdash_{cp} \lambda^L x.e : \tau \rightarrow^l \tau'}$$

By induction, $C; \Gamma\langle C\rangle, x : \tau\langle C\rangle \vdash_{cp} e' : \tau'\langle C\rangle$ and $C \vdash e \longrightarrow e'$. Then applying [Lam] yields $C; \Gamma\langle C\rangle \vdash_{cp} \lambda^L x.e' : (\tau \rightarrow^l \tau')\langle C\rangle$, and we also have $C \vdash \lambda^L x.e \longrightarrow \lambda^L x.e'$.

Case [App]. In this case, the term is $e_1 @^L e_2$, and there are three possible reductions. In the first case, when $e_1 @^L e_2 \longrightarrow e_1' @^L e_2$, we have

$$\text{[App]}\frac{\begin{array}{c}C; \Gamma \vdash_{cp} e_1 : \tau_2 \rightarrow^l \tau_1 \\ C; \Gamma \vdash_{cp} e_2 : \tau_2 \qquad C \vdash l \leq L\end{array}}{C; \Gamma \vdash_{cp} e_1 @^L e_2 : \tau_1}$$

Then by induction, $C; \Gamma\langle C\rangle \vdash_{cp} e_1' : (\tau_2 \rightarrow^l \tau_1)\langle C\rangle$. By Lemma 9, $C; \Gamma\langle C\rangle \vdash_{cp} e_2 : \tau_2\langle C\rangle$. Thus we can apply [App] to yield $C; \Gamma\langle C\rangle \vdash_{cp} e_1' @^L e_2 : \tau_1\langle C\rangle$. Also by induction, $C \vdash e_1 \longrightarrow e_1'$, so $C \vdash e_1 @^L e_2 \longrightarrow e_1' @^L e_2$. The second case, when $e_1 @^L e_2 \longrightarrow e_1 @^L e_2'$, is similar.

In the last case, we have $(\lambda^L x.e_1)@^{L'}e_2 \longrightarrow e_1[x \mapsto e_2]$. In this case, we have

$$\dfrac{\dfrac{\dfrac{C;\Gamma, x:\tau_1 \vdash e_1:\tau_2 \quad C \vdash L \le l'}{C;\Gamma \vdash_{cp} \lambda^L x.e_1 : \tau_1 \to^{l'} \tau_2}\,[\text{Lam}] \qquad C;\emptyset \vdash (\tau_1 \to^{l'} \tau_2) \le (\tau' \to^{l} \tau)}{C;\Gamma \vdash_{cp} \lambda^L x.e_1 : \tau' \to^l \tau}\,[\text{Sub}] \qquad C;\Gamma \vdash_{cp} e_2 : \tau' \qquad C \vdash l \le L'}{C;\Gamma \vdash (\lambda^L x.e_1)@^{L'}e_2 : \tau}\,[\text{App}]$$

Then $C;\emptyset \vdash \tau' \le \tau_1$, hence $C;\Gamma \vdash_{cp} e_2 : \tau_1$. Then by Lemma 10, $C;\Gamma\langle C\rangle \vdash_{cp} e_1[x \mapsto e_2] : \tau_2\langle C\rangle$. By Lemma 8, $C;\emptyset \vdash \tau_2\langle C\rangle \le \tau\langle C\rangle$. Thus by [Sub] we have $C;\Gamma\langle C\rangle \vdash_{cp} e_1[x \mapsto e_2] : \tau\langle C\rangle$.

Finally, in this reduction step L' consumes L. But $C \vdash L \le l'$, $C \vdash l' \le l$, and $C \vdash l \le L'$, hence $C \vdash L \le L'$. Hence we have shown the conclusion.

Case [Pair]. In this case, we have either $(e_1,e_2)^L \longrightarrow (e_1',e_2)^L$ or $(e_1,e_2)^L \longrightarrow (e_1,e_2')^L$. In either case the proof proceeds by induction, similar to the first case of [App].

Case [Proj]. In this case, the term is $e.^L j$. There are two possible reductions. If the reduction is $e.^L j \longrightarrow e'.^L j$, then we apply induction as in the first case of [App]. Otherwise, the reduction is $(e_1,e_2)^{L'}.^L j \longrightarrow e_j$. In this case, our typing proof is of the form

$$\dfrac{\dfrac{\dfrac{C;\Gamma \vdash e_1 : \tau_1' \quad C;\Gamma \vdash e_2 : \tau_2' \quad C \vdash L' \le l'}{C;\Gamma \vdash_{cp} (e_1,e_2)^{L'} : \tau_1' \times^{l'} \tau_2'}\,[\text{Pair}] \qquad C;\emptyset \vdash \tau_1' \times^{l'} \tau_2' \le \tau_1 \times^l \tau_2}{C;\Gamma \vdash_{cp} (e_1,e_2)^{L'} : \tau_1 \times^l \tau_2}\,[\text{Sub}] \qquad C \vdash l \le L \qquad j = 1,2}{C;\Gamma \vdash_{cp} (e_1,e_2)^{L'}.^L j : \tau_j}\,[\text{Proj}]$$

Then $C;\Gamma \vdash e_j : \tau_j'$, and since $C;\emptyset \vdash \tau_j' \le \tau_j$, we have $C;\Gamma \vdash e_j : \tau_j$. Then by Lemma 9, we have $C;\Gamma\langle C\rangle \vdash e_j : \tau_j\langle C\rangle$. Also, $C \vdash L' \le l'$, $C \vdash l' \le l$, and $C \vdash l \le L$, hence $C \vdash L' \le L$, and we have shown the conclusion.

Case [Cond]. In this case, the term is $\texttt{if0}^L\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$, and there are four possible reductions. If the reduction occurs inside of e_0, e_1, or e_2, then we proceed by induction as usual. Otherwise, the reduction is either $\texttt{if0}^L\ n^{L'}\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_1$ or $\texttt{if0}^L\ n^{L'}\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \longrightarrow e_2$, depending on whether n is 0. In either case, our typing judgment looks like

$$\dfrac{\dfrac{\dfrac{C \vdash L' \le l'}{C;\Gamma \vdash_{cp} n^{L'} : int^{l'}}\,[\text{Int}] \qquad C;\emptyset \vdash int^{l'} \le int^l}{C;\Gamma \vdash_{cp} n^{L'} : int^l}\,[\text{Sub}] \qquad C \vdash l \le L \quad C;\Gamma \vdash_{cp} e_1 : \tau \quad C;\Gamma \vdash_{cp} e_2 : \tau}{C;\Gamma \vdash_{cp} \texttt{if0}^L\ n^{L'}\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau}\,[\text{Cond}]$$

Then clearly $C;\Gamma \vdash_{cp} e_i : \tau$, and by Lemma 9 we have $C;\Gamma\langle C\rangle \vdash_{cp} e_i : \tau\langle C\rangle$. And since $C \vdash L' \leq l'$, $C \vdash l' \leq l$, and $C \vdash l \leq L$, we have $C \vdash L' \leq L$.

Case [Sub]. In this case, the reduction is $e \longrightarrow e'$, and we have

$$\text{[Sub]}\frac{C;\Gamma \vdash_{cp} e : \tau_1 \qquad C;\emptyset \vdash \tau_1 \leq \tau_2}{C;\Gamma \vdash_{cp} e : \tau_2}$$

But by induction, $C;\Gamma\langle C\rangle \vdash_{cp} e' : \tau_1\langle C\rangle$. By Lemma 8, $C;\emptyset \vdash \tau_1\langle C\rangle \leq \tau_2\langle C\rangle$. Hence we can apply [Sub] to show $C;\Gamma\langle C\rangle \vdash_{cp} e' : \tau_2\langle C\rangle$.

Case [Let]. In this case, the term is $\texttt{let}\ f = e_1\ \texttt{in}\ e_2$. If the reduction occurs inside of e_1 or e_2, then we proceed by induction as usual. Otherwise, the typing judgment is of the form

$$\text{[Let]}\frac{\begin{array}{c} C';\Gamma \vdash_{cp} e_1 : \tau_1 \qquad C;\Gamma, f : \forall\vec{\alpha}[C'].\tau_1 \vdash_{cp} e_2 : \tau_2 \\ \vec{\alpha} \subseteq (fl(\tau_1) \cup fl(C')) \setminus fl(\Gamma) \end{array}}{C;\Gamma \vdash_{cp} \texttt{let}\ f = e_1\ \texttt{in}\ e_2 : \tau_2}$$

and the reduction is $\texttt{let}\ f = e_1\ \texttt{in}\ e_2 \longrightarrow e_2[f \mapsto e_1]$. But then by Lemma 11, $C;\Gamma \vdash e_2[f \mapsto e_1] : \tau_2$. Then by Lemma 9, $C;\Gamma\langle C\rangle \vdash e_2[f \mapsto e_1] : \tau_2\langle C\rangle$. Since no labeled values are consumed by this reduction, $C \vdash \texttt{let}\ f = e_1\ \texttt{in}\ e_2 \longrightarrow e_2[f \mapsto e_1]$ trivially, and we are done.

Case [Fix]. Analogous to [Let].

Case [Inst]. This case cannot happen, because we assume $e \longrightarrow e'$.

Case [Pack]. This case proceeds by induction as usual. In this case the reduction must be $\texttt{pack}^{L,i}\ e \longrightarrow \texttt{pack}^{L,i}\ e'$, and so we proceed by the usual induction. The typing proof is

$$\text{[Pack]}\frac{C;\Gamma \vdash_{cp} e : \tau[\vec{\alpha} \mapsto \vec{l}] \qquad C \vdash C'[\vec{\alpha} \mapsto \vec{l}] \qquad C \vdash L \leq l}{C;\Gamma \vdash_{cp} \texttt{pack}^{L,i}\ e : \exists^l \vec{\alpha}[C'].\tau}$$

Then by induction, $C;\Gamma\langle C\rangle \vdash_{cp} e' : (\tau[\vec{\alpha} \mapsto \vec{l}])\langle C\rangle$ and $C \vdash e \longrightarrow e'$. By our alpha renaming convention, $(\tau[\vec{\alpha} \mapsto \vec{l}])\langle C\rangle = (\tau\langle C\rangle)[\vec{\alpha} \mapsto \vec{l}]$. Further, $C \vdash C \cup C'[\vec{\alpha} \mapsto \vec{l}]$, and again by our alpha renaming convention $C \vdash (C \cup C')[\vec{\alpha} \mapsto \vec{l}]$. Hence by [Pack] we have $C;\Gamma\langle C\rangle \vdash_{cp} \texttt{pack}^{L,i}\ e : (\exists^l \vec{\alpha}[C'].\tau)\langle C\rangle$, and we also have $C \vdash \texttt{pack}^{L,i}\ e \longrightarrow \texttt{pack}^{L,i}\ e'$.

Case [Unpack]. In this case the term is $\texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2$, and there are three possible reductions. If the reudction occurs inside e_1 or e_2 then apply induction as usual. Otherwise, the reduction is $\texttt{unpack}^L\ x = (\texttt{pack}^{L',i}\ e)\ \texttt{in}\ e_2 \longrightarrow e_2[x \mapsto e]$, and the typing proof is

$$\text{[Unpack]}\frac{\begin{array}{c} \text{[Sub]}\dfrac{\text{[Pack]}\dfrac{\begin{array}{c} C;\Gamma \vdash_{cp} e : \tau_1[\vec{\alpha} \mapsto \vec{l}] \\ C \vdash C_1[\vec{\alpha} \mapsto \vec{l}] \qquad C \vdash L' \leq l_1 \end{array}}{C;\Gamma \vdash_{cp} \texttt{pack}^{L',i}\ e : \exists^{l_1}\vec{\alpha}[C_1].\tau_1} \quad C;\emptyset \vdash_{cp} \exists^{l_1}\vec{\alpha}[C_1].\tau_1 \leq \exists^{l_2}\vec{\alpha}[C_2].\tau_2}{C;\Gamma \vdash_{cp} \texttt{pack}^{L',i}\ e : \exists^{l_2}\vec{\alpha}[C_2].\tau_2} \\ C \vdash l_2 \leq L \qquad C \cup C_2;\Gamma, x : \tau_2 \vdash_{cp} e_2 : \tau \\ \vec{\alpha} \subseteq (fl(\tau_2) \cup fl(C_2)) \setminus (fl(\Gamma) \cup fl(\tau) \cup fl(C)) \end{array}}{C;\Gamma \vdash_{cp} \texttt{unpack}^L\ x = (\texttt{pack}^{L',i}\ e)\ \texttt{in}\ e_2 : \tau}$$

Further, the subtyping derivation is

$$[\text{Sub-}\exists]\,\frac{\begin{array}{c} C_1 \vdash C_2 \\ D' = [l \mapsto 1, \forall l \in \vec{\alpha}] \qquad C; D' \vdash \tau_1 \le \tau_2 \qquad C; D \vdash l_1 \le l_2 \end{array}}{C; \emptyset \vdash \exists^{l_1}\vec{\alpha}[C_1].\tau_1 \le \exists^{l_2}\vec{\alpha}[C_2].\tau_2}$$

We show soundness by applying the Substitution Lemma. First, we have $C \cup C_2; \Gamma, x : \tau_2 \vdash_{cp} e_2 : \tau$. Let ψ be the substitution $[\vec{\alpha} \mapsto \vec{l}]$. Applying this to our judgment yields $\psi(C) \cup \psi(C_2); \psi(\Gamma), x : \psi(\tau_2) \vdash_{cp} e_2 : \psi(\tau)$. But since $\vec{\alpha} \cap (fl(\Gamma) \cup fl(\tau) \cup fl(C)) = \emptyset$, we have $C \cup \psi(C_2); \Gamma, x : \psi(\tau_2) \vdash_{cp} e_2 : \tau$.

Further, since $C_1 \vdash C_2$, we have $\psi(C_1) \vdash \psi(C_2)$. Then since $C \vdash \psi(C_1)$, we have $C \vdash \psi(C_2)$. Then by Lemma 6, we have the following conclusion:

$$C; \Gamma, x : \psi(\tau_2) \vdash_{cp} e_2 : \tau$$

By assumption, we have $C; \Gamma \vdash_{cp} e : \psi(\tau_1)$. Also by assumption, we have $C; D' \vdash \tau_1 \le \tau_2$. Then by Lemma 7, we have $C; \emptyset \vdash \psi(\tau_1) \le \psi(\tau_2)$. Therefore by [Sub], we have the following conclusion:

$$C; \Gamma \vdash_{cp} e : \psi(\tau_2)$$

Now we can apply Lemma 10 to yield $C; \Gamma\langle C\rangle \vdash_{cp} e_2[x \mapsto e] : \tau\langle C\rangle$.

Finally, observe $C \vdash L' \le l_1$, $C \vdash l_1 \le l_2$, and $C \vdash l_2 \le L$. Hence $C \vdash L' \le L$, and therefore $C \vdash \texttt{unpack}^L\ x = (\texttt{pack}^{L',i}\ e)\ \texttt{in}\ e_2 \longrightarrow e_2[x \mapsto e]$, so we are done.

□

Theorem 14 (Soundness) *If $C; \Gamma \vdash_{cp} e : \tau$ and $e \longrightarrow^* e'$, then $C \vdash e \longrightarrow^* e'$.*

Proof: By induction on the length of the reduction $e \longrightarrow^* e'$, using Lemma 13.

□

B Reduction from CFL to Copy

In this section, we prove that typing proofs in CFL reduce to equivalent proofs in Copy. As mentioned earlier, CFL is actually more restrictive than Copy in the programs it is able to check.

We proceed following the basic proof technique in [26], but due to higher-order polymorphic types, our proof is somewhat more complicated.

Definition 15 (Polarity of label in type) *Let τ be a* CFL *type. We say that some label $l \in fl(\tau)$ has positive polarity (+) in τ if one of the following holds:*

1. $\tau = int^l$
2. $\tau = \tau_1 \rightarrow^{l'} \tau_2$ *and* $l = l'$ *or* l *has + polarity in* τ_2 *or* l *has − polarity in* τ_1.
3. $\tau = \tau_1 \times^{l'} \tau_2$ *and* $l = l'$ *or* l *has + polarity in* τ_1 *or in* τ_2.

4. $\tau = \exists^{l'}\vec{\alpha}.\tau'$ *and either* $l = l'$*, or* $l \notin \vec{\alpha}$ *and* l *has* $+$ *polarity in* τ'

Similarly, we say that some label $l \in fl(\tau)$ *has negative polarity* $-$ *in* τ *if one of the following holds:*

1. $\tau = \tau_1 \rightarrow^{l'} \tau_2$ *and* l *has* $-$ *polarity in* τ_2 *or* l *has* $+$ *polarity in* τ_1.
2. $\tau = \tau_1 \times^{l'} \tau_2$ *and* l *has* $-$ *polarity in* τ_1 *or in* τ_2.
3. $\tau = \tau = \exists^{l'}\vec{\alpha}.\tau'$ *and* $l \notin \vec{\alpha}$ *and* l *has* $-$ *polarity in* τ'.

Definition 16 (Polarized constraint sets) *Let* C *be a set of flow constraints, let* τ *be a* CFL *type and let* $F \subseteq fl(\tau)$*. We say that* C is p-polarized with respect to τ and F*, written* $C \rhd_F^p \tau$*, iff the following conditions hold for all* $l \in F$*:*

1. *whenever* $C \vdash l \leq l'$ *with* $l \neq l'$*, then* l *has polarity* $\bar{p}$ *in* τ.
2. *whenever* $C \vdash l' \leq l$ *with* $l \neq l'$*, then* l *has polarity* p *in* τ.

Lemma 17 *If* l *has polarity* p *in* τ*,* $I; D \vdash \tau \preceq_{p'}^i \tau' : \phi$*, and* $l \notin D$*, then* $I \vdash l \preceq_{p \cdot p'}^i \phi(l)$*, where* $p \cdot p = +$ *and* $p \cdot \bar{p} = -$*.*

Proof: Induction over the instantiation $I; D \vdash \tau \preceq_{p'}^i \tau' : \phi$:
Case [Inst-Int]. We have

$$\text{[Inst-Int]}\frac{I; D \vdash l \preceq_{p'}^i l' : \phi}{I; D \vdash int^l \preceq_{p'}^i int^{l'} : \phi}$$

Since $l \notin D$, we conclude that $I; D \vdash l \preceq_p^i l' : \phi$ can only have been proved by [Inst-Index-1]. Thus

$$\text{[Index-Index-1]}\frac{I \vdash l \preceq_{p'}^i l' \qquad \{(l, l')\} \in \phi}{I; \emptyset; \emptyset \vdash l \preceq_{p'}^i l' : \phi}$$

From this we get $l' = \phi(l)$ and $I \vdash l \preceq_{p'}^i \phi(l)$. But l has $+$ polarity in int^l. , and by definition, $p' \cdot + = p'$, and thus we have $I \vdash l \preceq_{p' \cdot +}^i \phi(l)$.

Case [Inst-Pair]. We have

$$\text{[Inst-Pair]}\frac{\begin{array}{c} I; D \vdash l_1 \preceq_{p'}^i l_2 : \phi \\ I; D \vdash \tau_1 \preceq_{p'}^i \tau_1' : \phi \qquad I; D \vdash \tau_2 \preceq_{p'}^i \tau_2' : \phi \end{array}}{I; D \vdash \tau_1 \times^{l_1} \tau_2 \preceq_{p'}^i \tau_1' \times^{l_2} \tau_2' : \phi}$$

There are two cases:

- $l = l_1$. Then as in the previous case, since $l \notin D$ we get $l_2 = \phi(l_1)$ and $I \vdash l \preceq_{p'}^i \phi(l)$ from [Inst-Index-1]. Then, since l has polarity $+$ in $\tau_1 \times^l \tau_2$, we have $I \vdash l \preceq_{p' \cdot +}^i \phi(l)$.
- $l \in fl(\tau_i)$ $(i = 1, 2)$. Then by Definition 15, l has polarity p in τ_i. Then by induction we have $I \vdash l \preceq_{p \cdot p'}^i \phi(l)$

Case [Inst-Fun]. We have

$$\text{[Inst-Fun]}\frac{\begin{array}{c}I; D \vdash l_1 \preceq^i_{p'} l_2 : \phi \\ I; D \vdash \tau_1 \preceq^i_{\bar{p'}} \tau'_1 : \phi \quad I; D \vdash \tau_2 \preceq^i_{p'} \tau'_2 : \phi\end{array}}{I; D \vdash \tau_1 \rightarrow^{l_1} \tau_2 \preceq^i_{p'} \tau'_1 \rightarrow^{l_2} \tau'_2 : \phi}$$

There are three cases:

- $l = l_1$. Then since $l \notin D$, as before by [Inst-Index-1] we have $l_2 = \phi(l_1)$ and $I \vdash l \preceq^i_{p'} \phi(l)$. Since l has polarity $+$ in $\tau_1 \rightarrow^l \tau_2$, we then have $I \vdash l \preceq^i_{p' \cdot +} \phi(l)$.
- $l \in fl(\tau_1)$. By Definition 15, l has polarity $\bar{p}$ in τ_1. Then by induction we have $I \vdash l \preceq^i_{\bar{p} \cdot \bar{p'}} \phi(l)$ But since $\bar{p} \cdot \bar{p'} = p \cdot p'$, this is equivalent to $I \vdash l \preceq^i_{p \cdot p'} \phi(l)$.
- $l \in fl(\tau_2)$. By Definition 15, l has polarity p in τ_2. As before, by induction we then have $I \vdash l \preceq^i_{p \cdot p'} \phi(l)$.

Case [Inst-$\exists$]. We have

$$\text{[Inst-}\exists\text{]}\frac{\begin{array}{c}D' = D \oplus \vec{\alpha} \quad I; D' \vdash \tau_1 \preceq^i_{p'} \tau_2 : \phi \\ I; D \vdash l_1 \preceq^i_{p'} l_2 : \phi\end{array}}{I; D \vdash \exists^{l_1} \vec{\alpha}.\tau_1 \preceq^i_{p'} \exists^{l_2} \vec{\alpha}.\tau_2 : \phi}$$

There are two cases:

- Case $l = l_1$. Then since $l \notin D$, by [Inst-Index-1] we have $l_2 = \phi(l_1)$ and $I \vdash l \preceq^i_{p'} \phi(l)$. Since l has polarity $+$ in $\exists^{l_1} \vec{\alpha}.\tau_1$, we then have $I \vdash l \preceq^i_{p' \cdot +} \phi(l)$.
- Case $l \in fl(\tau_1)$. We may assume $l \notin \vec{\alpha}$, since otherwise $l \notin fl(\exists^{l_1} \vec{\alpha}.\tau_1)$. Therefore $l \notin D'$. Further, by Definition 15 the polarity of l in τ_1 is p. Then by induction we have $I \vdash l \preceq^i_{p \cdot p'} \phi(l)$.

□

Definition 18 (Instantiation context) *Every application of an [Inst]*

$$\textit{[Inst]}\frac{\begin{array}{c}I; \emptyset \vdash \tau \preceq^i_+ \tau' : \phi \\ dom(\phi) = \vec{\alpha} \quad I \vdash \vec{l} \preceq^i_+ \vec{l} \quad I \vdash \vec{l} \preceq^i_- \vec{l}\end{array}}{I; C; \Gamma, f : (\forall \vec{\alpha}.\tau, \vec{l}) \vdash_{CFL} f^i : \tau'}$$

defines a positive instantiation context $\langle C, I, \vec{\alpha}, \vec{l}, \tau, \phi, +, i \rangle$.

Every application of [Pack]

$$\textit{[Pack]}\frac{\begin{array}{cc}I; C; \Gamma \vdash_{CFL} e : \tau' & I; \emptyset \vdash \tau \preceq^i_- \tau' : \phi \\ dom(\phi) = \vec{\alpha} & C \vdash L \leq l\end{array}}{I; C; \Gamma \vdash_{CFL} \texttt{pack}^{L,i}\ e : \exists^l \vec{\alpha}.\tau}$$

$$[\text{LubL}]\,\frac{C \vdash l_1 \le l \quad \cdots \quad C \vdash l_n \le l}{C \vdash (\bigsqcup_{i=1}^{n} l_i) \le l}$$

$$[\text{LubR}]\,\frac{j \in \{1, \ldots, n\}}{C \vdash l_j \le (\bigsqcup_{i=1}^{n} l_i)}$$

Figure 15: Extended Subtype Relation

defines a negative instantiation context $\langle C, I, \vec{\alpha}, \tau, \phi, -, i\rangle$

Since there is a unique i for every [Inst] or [Pack] rule, we define $InstCtx(i, D)$ to be the instantiation context defined at the rule identified by i in the CFL derivation $\mathcal{D}$.

Definition 19 (Closure) *Let C and I be CFL constraints. The we define the closure of the constraints as $C^I = \{l_1 \le l_2 \mid I; C \vdash l_1 \rightsquigarrow_m l_2\}$.*

Definition 20 *A set of instantiation constraints I is* normal *if whenever $I \vdash l_1 \preceq_p^i l_2$ and $I \vdash l_3 \preceq_{p'}^j l_4$ with $l_1 \ne l_2$ and $l_3 \ne l_4$, then $l_2 \ne l_3$.*

Definition 21 *A positive instantiation context $\langle C, I, \vec{\alpha}, \vec{l}, \tau, \phi, +, i\rangle$ is* normal *if*

1. $I; \emptyset \vdash \tau \preceq_p^i \tau' : \phi$
2. $\vec{\alpha} \cap \vec{l} = \emptyset$
3. $C \rhd_{\vec{\alpha}}^{+} \tau$
4. $I \vdash \vec{l} \preceq_{+}^{i} \vec{l}$ *and*
5. *I is normal*

Notice that by definition of $C \rhd_{\vec{\alpha}}^{p} \tau$ we also have $\vec{\alpha} \subseteq fl(\tau)$. We will define normal negative instantiation contexts after proving some important lemmas.

In order to show how derivations in CFL relate to derivations in Copy, we will need to relate types $(\forall\vec{\alpha}.\tau, \vec{l})$ with types $\forall\vec{\alpha}[C].\tau$, and similarly for existential types. However, notice that these types may be quantified over different variables—in the Copy type, we may quantify over variables appearing in τ and C, whereas in the CFL type we only explicitly quantify over variables appearing in τ. To make these match, we need to observe that if a variable appears in C and not in τ or $\vec{l}$, then it is an intermediate variable—the only thing that we really need to capture is how it induces constraints among variables that appear in τ. Hence we add to our system formal joins $(\bigsqcup_{i=1}^{n} l_i)$ of label variables. In the course of the proof, we will replace all intermediate variables in with joins among the variables in τ, the variables in $\vec{l}$, and constants. Figure 15 gives additional rules we use when checking $C \vdash l_1 \le l_2$ in addition to containment $\{l_1 \le l_2\} \in C$. For the remainder of this section, we write $C \vdash C'$ if for all $\{l_1 \le l_2\} \in C'$ we have $C \vdash l_1 \le l_2$.

Formally, we define $\Phi_S(l) = \{l' \in S \cup L \mid C^I \vdash l' \leq l\}$. For a set of labels S, we then define a substitution

$$\psi_S(l) = \begin{cases} l & l \in S \cup L \\ \bigsqcup \Phi_S(l) & \textit{otherwise} \end{cases}$$

Finally, for a set of labels S, we define $C^I_S = \psi_S(C^I)$, i.e., we replace labels in C^I that are not in S by the least-upper bound of labels in S that flow to it.

Lemma 22 *If $S \subseteq S'$, then $C^I_{S'} \vdash C^I_S$.*

Proof: Pick some $l \in S$. Then in C^I_S, there are two cases. Either l is mapped to itself, if $l \in S$, or l is mapped to $\bigsqcup \Phi_S(l)$. Now suppose $C^I \vdash l_1 \leq l_2$. If $l_1 \in S$ and $l_2 \in S$ then $C^I_S \vdash l_1 \leq l_2$ and $C^I_{S'} \vdash l_1 \leq l_2$ by the above reasoning. If $l_1 \notin S$ and $l_2 \notin S$, then we have $C^I_S \vdash \bigsqcup \Phi_S(l_1) \leq \bigsqcup \Phi_S(l_2)$. Then by [LubL] and [LubR], there exists an $l'_2 \in \Phi_S(l_2)$ such that for all $l'_1 \in \Phi_S(l_1)$, we have $C^I_S \vdash l'_1 \leq l'_2$, and notice that $l'_1, l'_2 \in S \subseteq S'$. Thus we have $C^I_{S'} \vdash l'_1 \leq l'_2$. But since this holds for all l'_1 and some l'_2, by [LubL] and [LubR] we have $C^I_{S'} \vdash \bigsqcup \Phi_S(l_1) \leq \bigsqcup \Phi_S(l_2)$. The reasoning for the other possibilities for l_1 and l_2 is similar.

□

Lemma 23 *If $S \subseteq S'$, then $\psi_S(C^I_{S'}) = C^I_S$.*

Lemma 24 *If $\langle C, I, \vec{\alpha}, \vec{l}, \tau, \phi, +, i\rangle$ is a normal positive instantiation context, then $C^I \rhd^p_{\vec{\alpha}} \tau$.*

Proof: We will show one case; the other polarity is similar. Suppose $C^I \vdash l \leq l'$ with $l \neq l'$ and $l \in \vec{\alpha}$. Then we have $I; C \vdash l \rightsquigarrow_m l'$. We need to show that l has polarity $\bar{p}$ in τ. The proof is by induction on the derivation of $I; C \vdash l \rightsquigarrow_m l'$.

Case [Level]. We have $C \vdash l \leq l'$. But then since the instantiation context is normal, $C \rhd^p_{\vec{\alpha}} \tau$, and thus l has polarity $\bar{p}$ in τ.

Case [Trans]. We have $I; C \vdash l \rightsquigarrow_m l''$ and $I; C \vdash l'' \rightsquigarrow_m l'$. By induction, $I; C \vdash l \rightsquigarrow_m l''$ implies that l has polarity $\bar{p}$ in τ.

Case [Constant]. This case cannot occur, because we assume $l \in \vec{\alpha}$.

Case [Match]. We have $I \vdash l_1 \preceq^i_- l$, $I; C \vdash l_1 \rightsquigarrow_m l_2$, and $I \vdash l_2 \preceq^i_+ l'$. Then suppose for a contradiction that $l_1 \neq l$. Since $l \in \vec{\alpha}$ and $\vec{\alpha} \cap \vec{l} = \emptyset$, we have $I \vdash l \preceq^i_{p'} \phi(l)$ with $\phi(l) \neq l$. Then since the instantiation context is normal, we have $l \neq l$, a contradiction. Thus $l_1 = l$. But then we have $I; C \vdash l \rightsquigarrow_m l_2$, and so by induction we have that l has polarity $\bar{p}$ in τ.

□

Intuitively, the following lemma shows that subsets of C^I are closed with respect to substitutions ϕ that correspond to instantiation constraints. In order to show this, we extend a substitution ϕ to a substitution $\hat{\phi}$, which is the same as ϕ except that intermediate variables are replaced by joins. We will use this lemma in proving correspondence between [Inst] and [Pack] rules of the two systems. Below we write L for the set of constant labels.

Given a normal positive instantiation context $\langle C, I, \vec{\alpha}, \vec{l}, \tau, \phi, +, i\rangle$, we define $\Phi_i = \Phi_{S_i}$ where $S_i = \vec{\alpha} \cup \vec{l} \cup L$.

Lemma 25 *Let $\langle C, I, \vec{\alpha}, \vec{l}, \tau, \phi, +, i\rangle$ be a normal positive instantiation context. If $\vec{\alpha} \cup \vec{l} \subseteq S'$ and $\phi(S') \subseteq S$, then $C^I_S \vdash \hat{\phi}(C^I_{S'})$, where*

$$\hat{\phi}(l) = \begin{cases} \phi(l) & l \in \vec{\alpha} \\ l & l \in \vec{l} \cup L \\ \bigsqcup \hat{\phi}(\Phi_i(l)) & \textit{otherwise} \end{cases}$$

Proof: Suppose $\{l'_1 \leq l'_2\} \in \hat{\phi}(C^I_{S'})$. Then there are $l_1, l_2 \in S'$ such that $\{l_1 \leq l_2\} \in C^I_{S'}$, where $\hat{\phi}(l_1) = l'_1$ and $\hat{\phi}(l_2) = l'_2$, and thus $C^I \vdash l_1 \leq l_2$ by Lemma 22. Notice we can assume $l_1 \neq l_2$, since otherwise the proof is trivial. We can also assume without loss of generality that neither l_1 nor l_2 is a join, because if it is, we can use [LubL] and [LubR] to reduce the inequality to a set of inequalities among labels, as in Lemma 22. So then we need to show $C^I_S \vdash l'_1 \leq l'_2$. There are nine possible cases, depending on where each of $l \in \{l_1, l_2\}$ appears:

1. $l \in \vec{l} \cup L$, and so $\hat{\phi}(l) = l$
2. $l \in \vec{\alpha}$, and so $\hat{\phi}(l) = \phi(l)$ (this is disjoint from the first case since the instantiation context is normal)
3. otherwise $\hat{\phi}(l) = \bigsqcup \hat{\phi}(\Phi_i(l))$

We proceed by case analysis.

1. $l_1 \in \vec{l} \cup L$ and $l'_1 = \hat{\phi}(l_1) = l_1$. The cases for l_2 are:

   (a) $l_2 \in \vec{l} \cup L$ and $l'_2 = \hat{\phi}(l_2) = l_2$. Then since $C^I \vdash l_1 \leq l_2$ and $l'_i = l_i$, we have $C^I \vdash l'_1 \leq l_2$. And since $\phi(S') \subseteq S$, we have $l'_1, l'_2 \in S$. Thus $C^I_S \vdash l'_1 \leq l'_2$.

   (b) $l_2 \in \vec{\alpha}$ and $l'_2 = \hat{\phi}(l_2) = \phi(l_2)$. Then since the instantiation context is normal, we have $C \rhd^p_{\vec{\alpha}} \tau$. Then by Lemma 24 we have $C^I \rhd^p_{\vec{\alpha}} \tau$. But then since $C^I \vdash l_1 \leq l_2$ and $l_2 \in \vec{\alpha}$, we know l_2 has polarity p in τ. Then since $l_2 \notin \vec{l}$, by Lemma 17 we have $I \vdash l_2 \preceq^i_+ \phi(l_2)$, since $p \cdot p = +$ and in the instantiation $D = \emptyset$. Since the instantiation context is normal, we either have $l_1 \in \vec{l}$ or $l_1 \in L$, all of which imply $I \vdash l_1 \preceq^i_\pm l_1$ (the former by [Inst] or [Pack], and the latter by [Constant]). Then by [Match], we have

   $$[\text{Match}]\frac{I \vdash l_1 \preceq^i_- \phi(l_1) \quad I; C \vdash l_1 \rightsquigarrow_m l_2 \quad I \vdash l_2 \preceq^i_+ \phi(l_2)}{I; C \vdash \phi(l_1) \rightsquigarrow_m \phi(l_2)}$$

   and so $C^I \vdash l'_1 \leq l'_2$. Since $\phi(S') \subseteq S$, we have $l'_1, l'_2 \in S$. Thus $C^I_S \vdash l'_1 \leq l'_2$.

   (c) Otherwise $l'_2 = \hat{\phi}(l_2) = \bigsqcup \hat{\phi}(\Phi_i(l_2))$ Then since $C^I \vdash l_1 \leq l_2$ and either $l_1 \in \vec{l}$, or $l_1 \in L$, we have $l_1 \in \Phi_i(l_2)$. Since $\hat{\phi}(l_1) = l_1$, we have $l_1 \in \hat{\phi}(\Phi_i(l_2))$. But then from [LubR], we get $C^I \vdash l_1 \leq \hat{\phi}(l_2) = \bigsqcup \hat{\phi}(\Phi_i(l_2))$. And since $\phi(S') \subseteq S$, we have $l'_1 \in S$ and thus $C^I_S \vdash l'_1 \leq l'_2$.

2. $l_1 \in \vec{\alpha}$. Then $l'_1 = \hat{\phi}(l_1) = \phi(l_1)$. The cases for l_2 are:

   (a) $l_2 \in \vec{l} \cup L$ and $\hat{\phi}(l_2) = l_2$. This is analogous to case 1(b). Since the instantiation context is normal, we have $C \rhd^p_{\vec{\alpha}} \tau$. Then by Lemma 24, we have $C^I \rhd^p_{\vec{\alpha}} \tau$. But since $C^I \vdash l_1 \leq l_2$

and $l_1 \in \vec{\alpha}$, we know that l_1 has polarity $\bar{p}$ in τ. Then since $l_1 \notin \vec{l}$, Then by Lemma 17, we have $I \vdash l_1 \preceq^i_- \phi(l_1)$, since $\bar{p} \cdot p = -$ and in the instantiation $D = \emptyset$. Since the instantiation context is normal and either $l_2 \in \vec{l}$ or $l_2 \in L$, we also have $I \vdash l_2 \preceq^i_\pm l_2$. Then by [Match], we have

$$[\text{Match}]\frac{I \vdash l_1 \preceq^i_- \hat{\phi}(l_1) \quad I; C \vdash l_1 \rightsquigarrow_m l_2 \quad I \vdash l_2 \preceq^i_+ \hat{\phi}(l_2)}{I; C \vdash \hat{\phi}(l_1) \rightsquigarrow_m \hat{\phi}(l_2)}$$

and so $C^I \vdash l'_1 \leq l'_2$. And since $\phi(S') \subseteq S$, we have $l'_1, l'_2 \in S$. Thus $C^I_S \vdash l'_1 \leq l'_2$.

(b) $l_2 \in \vec{\alpha}$ and $\hat{\phi}(l_2) = \phi(l_2)$. As in 2(a) above, we have $C^I \rhd^p_{\vec{\alpha}} \tau$. Since $C^I \vdash l_1 \leq l_2$, $l_1 \in \vec{\alpha}$, $l_1 \notin \vec{l}$, and $l_2 \in \vec{\alpha}$, we know that l_1 has polarity $\bar{p}$ in τ and l_2 has polarity p in τ, and in the instantiation $D = \emptyset$. Then by Lemma 17, we have $I \vdash l_1 \preceq^i_- l'_1$ and $I \vdash l_2 \preceq^i_+ l'_2$. Then we have

$$[\text{Match}]\frac{I \vdash l_1 \preceq^i_- \hat{\phi}(l_1) \quad I; C \vdash l_1 \rightsquigarrow_m l_2 \quad I \vdash l_2 \preceq^i_+ \hat{\phi}(l_2)}{I; C \vdash \hat{\phi}(l_1) \rightsquigarrow_m \hat{\phi}(l_2)}$$

so $C^I \vdash l'_1 \leq l'_2$. And since $\phi(S') \subseteq S$, we have $l'_1, l'_2 \in S$. Thus $C^I_S \vdash l'_1 \leq l'_2$.

(c) Otherwise $l'_2 = \hat{\phi}(l_2) = \bigsqcup \hat{\phi}(\Phi_i(l_2))$. Then since $C^I \vdash l_1 \leq l_2$ and $l_1 \in \vec{\alpha}$, we have $l_1 \in \Phi_i(l_2)$. So then $l'_1 = \hat{\phi}(l_1) \in \hat{\phi}(\Phi_i(l_2))$. Then from [LubR] we have $C^I \vdash l'_1 \leq \bigsqcup \hat{\phi}(\Phi_i(l_2))$. And since $\phi(S') \subseteq S$, we have $l'_1 \in S$. Therefore $C^I_S \vdash l'_1 \leq l'_2$.

3. Otherwise, $l'_1 = \hat{\phi}(l_1) = \bigsqcup \hat{\phi}(\Phi_i(l_1))$. The cases for l_2 are:

   (a) $l_2 \in \vec{l} \cup L$ and $\hat{\phi}(l_2) = \phi(l_2) = l_2$. Then since $C^I \vdash l_1 \leq l_2$, we have $C^I \vdash l' \leq l_2$ for all $l' \in \Phi_i(l_1)$ (and $l' \in S'$ by assumption) by [Trans] and [LubL] in Figure 14. Moreover, for each $l' \in \Phi_i(l_1)$, there are two cases.

      i. $l' \in \vec{l} \cup L$. Apply case 1(a) to show $C^I_S \vdash \hat{\phi}(l') \leq l'_2$.

      ii. $l' \in \vec{\alpha}$. Apply case 2(a) to show $C^I_S \vdash \hat{\phi}(l') \leq l'_2$.

   Thus for all $l' \in \Phi_i(l_1)$, we have $C^I_S \vdash \hat{\phi}(l') \leq l'_2$. Then by [LubL], we have $C^I_S \vdash \bigsqcup \hat{\phi}(\Phi_i(l_1)) \leq \hat{\phi}(l_2)$, or $C^I_S \vdash l'_1 \leq l'_2$.

   (b) $l_2 \in \vec{\alpha}$ and $\hat{\phi}(l_2) = \phi(l_2)$. Since $C^I \vdash l_1 \leq l_2$, we have $C^I \vdash l' \leq l_2$ for all $l' \in \Phi_i(l_1)$ by [Trans] (and $l' \in S'$ by assumption). For each $l' \in \Phi_i(l_1)$, there are two cases. If $l' \in \vec{l} \cup L$, apply case 1(b) to show $C^I_S \vdash \hat{\phi}(l') \leq l'_2$. If $l' \in \vec{\alpha}$, apply case 2(b) to show $C^I_S \vdash \hat{\phi}(l') \leq l'_2$. Then by [LubL] as before, $C^I_S \vdash l'_1 \leq l'_2$.

   (c) Otherwise $l'_2 = \hat{\phi}(l_2) = \bigsqcup \hat{\phi}(\Phi_i(l_2))$. Then since $C^I \vdash l_1 \leq l_2$, we have $C^I \vdash l' \leq l_2$ for all $l' \in \Phi_i(l_1)$ by [Trans]. But then $\Phi_i(l_1) \subseteq \Phi_i(l_2) \subseteq S'$. Therefore $\hat{\phi}(\Phi_i(l_1)) \subseteq \hat{\phi}(\Phi_i(l_2)) \subseteq S$. Then by [LubR] we have $C^I \vdash l' \leq (\bigsqcup \hat{\phi}(\Phi_i(l_2)))$ for all $l' \in \hat{\phi}(\Phi_i(l_1))$, and so by [LubL] we have $C^I \vdash (\bigsqcup \hat{\phi}(\Phi_i(l_1))) \leq (\bigsqcup \hat{\phi}(\Phi_i(l_2)))$. Since $\hat{\phi}(\Phi_i(l_1)) \subseteq \hat{\phi}(\Phi_i(l_2)) \subseteq S$, we have $C^I_S \vdash l'_1 \leq l'_2$.

$\square$

Definition 26 *A negative instantiation context $\langle C, I, \vec{\alpha}, \tau, \phi, -, i\rangle$ is* normal *if*

1. $I; \emptyset \vdash \tau \preceq^i_p \tau' : \phi$

2. $C \rhd^-_{\vec{\alpha}} \tau$

3. $fl(\tau') \cap \vec{\alpha} = \emptyset$

4. I *is normal*

Next we define a notion of a *normal* CFL *derivation*, which intuitively is one that corresponds directly to a derivation in Copy.

Definition 27 (Normal CFL derivation) *A CFL derivation $\mathcal{D}$ is* normal *if*

1. *Every instantiation context $InstCtx(i, D)$ is normal*

2. *For all universal types $(\forall\vec{\alpha}.\tau, \vec{l})$, it is the case that $\vec{\alpha} = fl(\tau) \setminus \vec{l}$.*

3. *For all existential types $\exists^{\vec{l}}\vec{\alpha}.\tau l$, it is the case that $C^I_{\vec{\alpha}} \rhd^-_{\vec{\alpha}} \tau$, i.e., the constraint sets in translated existential types are always negatively polarized with respect to the base type.*

4. *All polymorphic types created in [Let] and [Pack] have distinct bound labels $\vec{\alpha}$.*

5. *For every two sub-derivations $\mathcal{D}_1$, $\mathcal{D}_2$ in $\mathcal{D}$, where $\mathcal{D}_1$ is not a part of $\mathcal{D}_2$ and conversely, the only common labels between $\mathcal{D}_1$ and $\mathcal{D}_2$ are in the Γ assumptions and concluding types of $\mathcal{D}_1$ and $\mathcal{D}_2$.*

Notice that every sub-derivation of a normal CFL derivation is normal.

Lemma 28 *If $I; C; \Gamma \vdash_{CFL} e : \tau$, then there exists a normal CFL derivation $I'; C'; \Gamma \vdash_{CFL} e : \tau$.*

Proof: We walk through the conditions. Satisfying conditions 4 and 5 is a matter of picking fresh labels wherever possible. Condition 2 is satisfied by construction of [Let] and [Fix]. And condition 1 follows by reasoning similar to [26]. Observe that reasoning similar to Lemma 33 below shows that $C \rhd^p_{\vec{\alpha}} \tau$ at uses of [Pack].

The only tricky condition to show is 3. We sketch the proof. Consider the constraints generated in the e_2 sub-derivation portion of [Unpack]:

$$I; C; \Gamma, x : \tau \vdash_{CFL} e_2 : \tau'$$

Within the body of e_2, we can assume that [Sub] is always applied after x. Thus for any l appearing positively in τ, we will only generate constraints $l \leq l'$, and vice-versa for labels appearing negatively. Thus the constraints generated in this portion of the derivation are negatively polarized with respect to τ and $\vec{\alpha}$, and so far $C^I_{\vec{\alpha}} \rhd^-_{\vec{\alpha}} \tau$, since transitively closing these constraints does not affect polarity, and neither does restricting to $\vec{\alpha}$.

Otherwise, suppose we have an application of [Sub] with

$$C; \emptyset; \emptyset \vdash \exists^{l'_1}\vec{\alpha_1}.\tau_1 \leq \exists^{l'_2}\vec{\alpha_2}.\tau_2$$

Then let $l_i \in \vec{\alpha_i}$ be labels in the same positions in τ_i. Each occurs with the same polarity. Suppose the l_i appear positively. Then [Sub-Index-2] generates the constraint $C \vdash l_1 \leq l_2$. Clearly we have

$$
\begin{array}{rcl}
\Psi_{C,I}(int^l) & = & int^l \\
\Psi_{C,I}(\tau_1 \rightarrow^l \tau_2) & = & \Psi_{C,I}(\tau_1) \rightarrow^l \Psi_{C,I}(\tau_2) \\
\Psi_{C,I}(\tau_1 \times^l \tau_2) & = & \Psi_{C,I}(\tau_1) \times^l \Psi_{C,I}(\tau_2) \\
\Psi_{C,I}(\exists^l \vec{\alpha}.\tau) & = & \exists^l \vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau)) \\
\Psi_{C,I}(\Gamma, f : (\forall \vec{\alpha}.\tau, \vec{l})) & = & \Psi_{C,I}(\Gamma), f : \forall \vec{\alpha}[C^I_{(\vec{\alpha} \cup \vec{l})}].(\Psi_{C,I}(\tau)) \\
\Psi_{C,I}(\Gamma, x : \tau) & = & \Psi_{C,I}(\Gamma), (\Psi_{C,I}(\tau))
\end{array}
$$

Figure 16: Translation from CFL types to Copy types

violated the polarity restriction for l_2 in C. However, observe that in $C^I_{\vec{\alpha_2}}$, we have that l_1 is the join of no elements, and this holds transitively even with more applications of [Sub], since they can only add lower bounds to l_2 that do not appear in $\vec{\alpha_2}$. (Only [Unpack] can add constraints in the other direction, and once we unpack something we cannot re-pack it in the same scope). Thus this constraint is vacuous, and we ignore it for computing polarities. (If we did not ignore these constraints, then [LubL] would allow us to put any label on the right hand side of a constraint, in any constraint system.) Similarly, if the l_i appear negatively, [Sub] generates the constraint $C \vdash l_2 \leq l_1$, but in $C^I_{\vec{\alpha_2}}$, we have that l_1 is the join of some elements including l_2, which is again vacuous, and more applications of [Sub] can only add upper bounds to l_2 that do not appear in $\vec{\alpha_2}$.

Otherwise, suppose we have an application of [Inst] with

$$I; \emptyset \vdash \exists^{l'_1} \vec{\alpha}.\tau_1 \preceq^i_+ \exists^{l'_2} \vec{\alpha}.\tau_2$$

Then [Inst-Index-2] generates no constraints, and reasoning about the type $\exists^{l'_1}\vec{\alpha}.\tau_1$ shows that positively occurring labels in τ_1 can only have lower bounds and negatively occurring labels can only have upper bounds.

Finally, otherwise suppose we have an application of [Pack] with

$$I; \emptyset \vdash \exists^{l'_2} \vec{\alpha}.\tau_2 \preceq^i_- \exists^{l'_1} \vec{\alpha}.\tau_1$$

Then [Inst-Index-2] generates no constraints, and reasoning about the type $\exists^{l'_1}\vec{\alpha}.\tau_1$ shows that positively occurring labels in τ_1 can only have upper bounds and negatively occurring labels can only have lower bounds.

The cases of uses of [Sub], [Pack], and [Inst] deeper in a type are similar.

□

Finally, we can prove that for every normal CFL derivation, there exists an equivalent Copy derivation. Intuitively, a program type checks under CFL constraints I and C, then it should type check under Copy with constraints C^I (this turns out not quite to work; see below). When translating the derivation, we also need to choose the constraint systems for polymorphic Copy types, and these systems are implicit in CFL. Rehof, Fahndrich, and Das [26] choose C^I as the constraint system for all polymorphic types. However, this does not work in our system, because existentials are higher-order. We could translate the type $(\forall\vec{\alpha}.(\exists^{l'}\vec{\alpha'}.\tau), \vec{l})$ to $\forall\vec{\beta}[C^I].(\exists^{l'}\vec{\beta'}[C^I].\tau)$, but when we instantiate the latter, the instantiation might cause substitutions on some of the variables in the C^I of the existential type. Instead, for existentials, we put in a subset of C^I that is restricted to the bound labels in the type. By construction of the CFL system, these bound labels can never mix with free labels. Similarly, for universal types, we plug in C^I restricted to the bound labels and the free labels of the universal; for universals, free labels do not cause problems, because they are not first-class. Figure 16 defines a translation function $\Psi_{C,I}$ that takes COPY types and transforms them to CFL types. For an existential $\exists^l \vec{\alpha}.\tau$, we choose as the COPY constraints $C^I_{\vec{\alpha}}$. The strong hypothesis in [Unpack] in Figure 10 guarantees that this is safe, because quantified labels can never mix with non-quantified labels. For universal types, on the other hand, we allow quantified types to be constrained by non-quantified types, and thus for a type $(\forall \vec{\alpha}.\tau, \vec{l})$ we choose the constraints $C^I_{(\vec{\alpha}\cup\vec{l})}$. Intuitively, these are exactly the labels that "matter" to a caller of the quantified type—those that are bound in the type and those that may be free in the type. Any other labels (for example, intermediate labels constructed in the function body) are irrelevant except for their effects on $\vec{\alpha}$ and $\vec{l}$.

Lemma 29 *For any substitution ϕ, we have $\phi(\Psi_{C,I}(\tau)) = \Psi_{C,I}(\phi(\tau))$.*

Proof: By induction on the definition of $\Psi_{C,I}$. The interesting cases are existentials and universals. Letting $\phi'(l) = l$ for $l \in \vec{\alpha}$ and $\phi'(l) = \phi(l)$ otherwise, we have

$$\begin{array}{rcll} \phi(\Psi_{C,I}(\exists^l \vec{\alpha}.\tau)) & = & \phi(\exists^l \vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau))) & \\ & = & \exists^{\phi(l)} \vec{\alpha}[\phi'(C^I_{\vec{\alpha}})].\phi'(\Psi_{C,I}(\tau))) & \\ & = & \exists^{\phi(l)} \vec{\alpha}[C^I_{\vec{\alpha}}].\phi'(\Psi_{C,I}(\tau))) & \text{by definition of } \phi' \\ & = & \exists^{\phi(l)} \vec{\alpha}[C^I_{\vec{\alpha}}].\Psi_{C,I}(\phi'(\tau))) & \text{by induction} \\ & = & \Psi_{C,I}(\phi(\exists^l \vec{\alpha}.\tau)) & \end{array}$$

□

Lemma 30 *For any set S, we have $\psi_{(fl(\Gamma)\cup S)}(\Psi_{C,I}(\Gamma)) = \Psi_{C,I}(\Gamma)$.*

Proof: The proof is by induction. Let $\psi = \psi_{(fl(\Gamma)\cup S)}$. For regular types τ in the range of Γ, we have $\psi(\Psi_{C,I}(\tau)) = \Psi_{C,I}(\psi(\tau))$ by Lemma 29. But since τ is in the range of Γ, $\psi(\tau) = \tau$.

For universals, let $\psi'(l) = l$ for $l \in \vec{\alpha}$ and $\psi'(l) = \psi(l)$ otherwise, and then we have

$$\begin{array}{rcll} \psi(\Psi_{C,I}((\forall \vec{\alpha}.\tau, \vec{l}))) & = & \psi(\forall \vec{\alpha}[C^I_{(\vec{\alpha}\cup\vec{l})}].(\Psi_{C,I}(\tau))) & \\ & = & \forall \vec{\alpha}[\psi'(C^I_{(\vec{\alpha}\cup\vec{l})})].\psi'(\Psi_{C,I}(\tau)) & \\ & = & \forall \vec{\alpha}[C^I_{(\vec{\alpha}\cup\vec{l})}].\psi'(\Psi_{C,I}(\tau)) & \text{Since } \vec{l} \in fl(\Gamma) \\ & = & \forall \vec{\alpha}[C^I_{(\vec{\alpha}\cup\vec{l})}].\Psi_{C,I}(\psi'(\tau)) & \text{by Lemma 29} \\ & = & \Psi_{C,I}(\psi((\forall \vec{\alpha}.\tau, \vec{l}))) & \end{array}$$

□

Lemma 31 $fl(\Psi_{C,I}(\tau)) = fl(\tau)$

Proof: By induction on the definition of $\Psi_{C,I}$. The only interesting case is $\Psi_{C,I}(\exists^l\vec{\alpha}.\tau) = \exists^l\vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau))$. By induction, we have $fl(\Psi_{C,I}(\tau)) = fl(\tau)$. Then observe that $fl(C^I_{\vec{\alpha}}) = \vec{\alpha}$. Thus $fl(\exists^l\vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau))) = \{l\}\cup((fl(\Psi_{C,I}(\tau))\cup fl(C^I_{\vec{\alpha}}))\setminus\vec{\alpha}) = \{l\}\cup((fl(\Psi_{C,I}(\tau))\setminus\vec{\alpha}) = \{l\}\cup(fl(\tau)\setminus\vec{\alpha}) = fl(\exists^l\vec{\alpha}.\tau)$.

□

Lemma 32 *Given types from a normal derivation,* $fl(\Psi_{C,I}(\Gamma)) \subseteq fl(\Gamma)$.

Proof: The interesting case (ignoring the environment and focusing on the $\forall$ type) is $\Psi_{C,I}((\forall\vec{\alpha}.\tau,\vec{l})) = \forall\vec{\alpha}[C^I_{(\vec{\alpha}\cup\vec{l})}].(\Psi_{C,I}(\tau))$. Since the derivation is normal, $\vec{\alpha} = fl(\tau)\setminus\vec{l}$. Thus

$$\begin{aligned}
fl(\forall\vec{\alpha}[C^I_{(\vec{\alpha}\cup\vec{l})}].(\Psi_{C,I}(\tau))) &= \Big(fl(C^I_{(\vec{\alpha}\cup\vec{l})}) \cup fl(\Psi_{C,I}(\tau))\Big)\setminus\vec{\alpha} \\
&\subseteq \Big(\vec{\alpha}\cup\vec{l}\cup fl(\tau)\Big)\setminus\vec{\alpha} \\
&= \vec{l}\cup(fl(\tau)\setminus\vec{\alpha}) \\
&= fl((\forall\vec{\alpha}.\tau,\vec{l}))
\end{aligned}$$

□

Lemma 33 *Given a type* $\exists^l\vec{\alpha}.\tau$ *from a normal* CFL *derivation, we have* $C^I_{\vec{\alpha}}\rhd^-_{\vec{\alpha}}\tau$, *i.e., the constraint sets in translated existential types are always negatively polarized with respect to the base type.*

Proof:

□

Lemma 34 *Let* $\langle C, I, \vec{\alpha}, \tau, \phi, -, i\rangle$ *be a negative instantiation context in a normal* CFL *derivation. If* $\vec{\alpha}\subseteq S'$ *and* $\phi(S')\subseteq S$, *then* $C^I_S \vdash \check{\phi}(C^I_{S'})$, *where*

$$\check{\phi}(l) = \begin{cases} \phi(l) & l\in\vec{\alpha} \\ l & l\in L \\ \bigsqcup\check{\phi}(\Phi_i(l)) & \textit{otherwise}\end{cases}$$

Proof: The proof is the same as in Lemma 25, observing that all $l\in\vec{\alpha}$ only appear on the left-hand side of an instantiation constraint, by [Inst-Index-2] and assumption that the derivation is normal.

□

In order to translate subtyping derivations, we also need to translate the D from Figure 11 into the D of Figure 7. We define $\Psi_{C,I}(\emptyset) = \emptyset$, and $\Psi_{C,I}(D\oplus\vec{\alpha}) = (\Psi_{C,I}(D))[l\mapsto\Psi_{C,I}(D)+1, \forall l\in\vec{\alpha}]$.

Lemma 35 *If* $C; D; D\vdash l\leq l'$ *then* $C^I_{(l,l')}; \Psi_{C,I}(D)\vdash_{cp} l\leq l'$.

Proof: By induction on $C; D; D \vdash l \leq l'$.

Case [Sub-Index-1]. We have

$$\text{[Sub-Index-1 (CFL)]}\frac{C \vdash l \leq l'}{C; \emptyset; \emptyset \vdash l \leq l'}$$

Then since $\Psi_{C,I}(\emptyset) = \emptyset$ and $\emptyset(l) = \emptyset(l') = 0$, we have

$$\text{[Sub-Label-1 (Copy)]}\frac{\begin{array}{c}(\Psi_{C,I}(D))(l) = (\Psi_{C,I}(D))(l') = 0 \\ C^I_{(l,l')} \vdash l \leq l'\end{array}}{C^I_{(l,l')}; \Psi_{C,I}(D) \vdash_{cp} l \leq l'}$$

Case [Sub-Index-2]. We have

$$\text{[Sub-Index-2 (CFL)]}\frac{C \vdash l_j \leq l_j}{C; D \oplus \{l_1, \ldots, l_n\}; D \oplus \{l_1, \ldots, l_n\} \vdash l_j \leq l_j}$$

Notice that by assumption both D_i's must be the same. Also, notice that $\Psi_{C,I}(D \oplus \{l_1, \ldots, l_n\})(l_j) > 0$ by definition.

$$\text{[Sub-Label-2 (Copy)]}\frac{(\Psi_{C,I}(D \oplus \{l_1, \ldots, l_n\}))(l) > 0}{C^I_{(l_j)}; \Psi_{C,I}(D \oplus \{l_1, \ldots, l_n\}) \vdash_{cp} l_j \leq l_j}$$

Case [Sub-Index-3]. We have

$$\text{[Sub-Index-3 (CFL)]}\frac{C; D; D \vdash l \leq l' \qquad l \neq l_i \qquad l' \neq l_j \qquad \forall i, j \in [1..n]}{C; D \oplus \{l_1, \ldots, l_n\}; D \oplus \{l_1, \ldots, l_n\} \vdash l \leq l'}$$

By induction, $C^I_{(l,l')}; \Psi_{C,I}(D) \vdash_{cp} l \leq l'$. Let $\vec{\alpha} = \{l_1, \ldots, l_n\}$. Then since $l, l' \neq l_i$ for any i, we have $(\Psi_{C,I}(D \oplus \vec{\alpha}))(l) = (\Psi_{C,I}(D))(l)$ and $(\Psi_{C,I}(D \oplus \vec{\alpha}))(l') = (\Psi_{C,I}(D))(l')$. Then there are two cases:

1. If $C^I_{(l,l')}; \Psi_{C,I}(D) \vdash_{cp} l \leq l'$ by [Sub-Label-1 (Copy)], then $(\Psi_{C,I}(D))(l) = (\Psi_{C,I}(D))(l') = 0$. But then $(\Psi_{C,I}(D \oplus \vec{\alpha})(l) = (\Psi_{C,I}(D \oplus \vec{\alpha})(l') = 0$, so we have

$$\text{[Sub-Label-1 (Copy)]}\frac{\begin{array}{c}(\Psi_{C,I}(D \oplus \vec{\alpha}))(l) = (\Psi_{C,I}(D \oplus \vec{\alpha}))(l') = 0 \\ C^I_{(l,l')} \vdash l \leq l'\end{array}}{C^I_{(l,l')}; \Psi_{C,I}(D \oplus \vec{\alpha}) \vdash_{cp} l \leq l'}$$

2. If $C^I_{(l,l')}; \Psi_{C,I}(D) \vdash_{cp} l \leq l'$ by [Sub-Label-2 (Copy)], then $(\Psi_{C,I}(D))(l) > 0$ and $l = l'$, and therefore $(\Psi_{C,I}(D \oplus \vec{\alpha}))(l) > 0$, and so we have

$$\text{[Sub-Label-2 (Copy)]}\frac{(\Psi_{C,I}(D \oplus \vec{\alpha}))(l) > 0}{C^I_{(l,l')}; \Psi_{C,I}(D \oplus \vec{\alpha}) \vdash_{cp} l \leq l'}$$

$\square$

Lemma 36 (Subtyping reduction from CFL to Copy) *Let $\mathcal{D}$ be a normal CFL derivation of $C; D; D \vdash \tau \leq \tau'$. Then $C^I_{(\tau,\tau')}; \Psi_{C,I}(D) \vdash_{cp} \Psi_{C,I}(\tau) \leq \Psi_{C,I}(\tau')$.*

Proof: By induction on the given CFL derivation.

Case [Sub-Int]. We have

$$[\text{Sub-Int (CFL)}]\frac{C; D; D \vdash l \leq l'}{C; D; D \vdash int^{l} \leq int^{l'}}$$

Then by Lemma 35 we have $C^I_{(l,l')}; \Psi_{C,I}(D) \vdash_{cp} l \leq l'$. But then we have

$$[\text{Sub-Int (Copy)}]\frac{C^I_{(l,l')}; \Psi_{C,I}(D) \vdash_{cp} l \leq l'}{C^I_{(l,l')}; \Psi_{C,I}(D) \vdash_{cp} int^{l} \leq int^{l'}}$$

Case [Sub-Pair], [Sub-Fun]. By induction, using Lemma 35 and the definition of $\Psi_{C,I}$.

Case [Sub-$\exists$]. We have

$$[\text{Sub-}\exists]\frac{\begin{array}{cc} D_1 = D \oplus \vec{\alpha_1} & D_2 = D \oplus \vec{\alpha_2} \\ C; D_1; D_2 \vdash \tau_1 \leq \tau_2 & C; D; D \vdash l_1 \leq l_2 \end{array}}{C; D; D \vdash \exists^{l_1}\vec{\alpha_1}.\tau_1 \leq \exists^{l_2}\vec{\alpha_2}.\tau_2}$$

Let $T = \{l_1, l_2\} \cup (fl(\tau_1) \setminus \vec{\alpha_1}) \cup (fl(\tau_2) \setminus \vec{\alpha_2})$. Let ϕ be an alpha-renaming such that $\phi(\vec{\alpha_2}) = \vec{\alpha_1}$. This is always well-defined by the assumption that the derivation is normal and by the subtyping rules of Figure 11. Then $\phi(D_2) = D_1$, and thus since $C; D_1; D_2 \vdash \tau_1 \leq \tau_2$, we have $C; D_1; D_1 \vdash \tau_1 \leq \phi(\tau_2)$. Then by induction we have $C^I_{(\tau_1,\phi(\tau_2))}; \Psi_{C,I}(D_1) \vdash_{cp} \Psi_{C,I}(\tau_1) \leq \Psi_{C,I}(\phi(\tau_2))$. But notice that by [Sub-Label-2] in Figure 7, no (nontrivial) constraints between variables in $\Psi_{C,I}(D_1)$ are ever generated. Thus we have $C^I_{((\tau_1,\phi(\tau_2))\setminus\vec{\alpha_1})}; \Psi_{C,I}(D_1) \vdash_{cp} \Psi_{C,I}(\tau_1) \leq \Psi_{C,I}(\phi(\tau_2))$. Notice that by definition of ϕ, we have

$$((fl(\tau_1) \setminus \vec{\alpha_1}) \cup (fl(\tau_2) \setminus \vec{\alpha_2})) = ((fl(\tau_1) \setminus \vec{\alpha_1}) \cup (\phi(fl(\tau_2)) \setminus \vec{\alpha_1}))$$

And thus by Lemmas 22 and 29 we have $C^I_T; \Psi_{C,I}(D_1) \vdash_{cp} \Psi_{C,I}(\tau_1) \leq \phi(\Psi_{C,I}(\tau_2))$.

Also by Lemmas 35 and 22 we have $C^I_T, \Psi_{C,I}(D) \vdash l_1 \leq l_2$.

We need to show that $C^I_{\vec{\alpha_1}} \vdash \phi(C^I_{\vec{\alpha_2}})$. Since the derivation is normal, we have $C^I_{\vec{\alpha_2}} \triangleright^-_{\vec{\alpha_2}} \tau_2$. Observe that by the subtyping rules in Figure 11, for label $l \in \vec{\alpha_2}$, if l has polarity $+$ in τ_2 then $\phi(l) \leq l$, and if l has polarity $-$ in τ_2 then $l \leq \phi(l)$.

Pick a label $l \in \vec{\alpha_2}$. Suppose that $C^I_{\vec{\alpha_2}} \vdash l' \leq l$. Then by definition, l has polarity $-$ in τ_2. Thus $l \leq \phi(l)$. By construction, l' is a join of the constants and labels in $\vec{\alpha_2}$, and by [LubL], we have that for all labels $l'' \in l'$ we have $C^I_{\vec{\alpha_2}} \vdash l'' \leq l'$. Then l'' has polarity $+$ in τ_2, and thus $\phi(l'') \leq l''$. But then $C^I \vdash \phi(l'') \leq \phi(l)$. And since this holds for all $l'' \in l'$, by [LubL] we have $C^I_{\vec{\alpha_1}} \vdash \phi(l'') \leq \phi(l)$, since $\phi(l''), \phi(l) \in \vec{\alpha_1}$.

Similarly, Suppose that $C^I_{\vec{\alpha_2}} \vdash l \leq l'$. Then by definition, l has polarity $+$ in τ_2, and hence $\phi(l) \leq l$. By construction, l' is a join of the constants and labels in $\vec{\alpha_2}$. There are two cases. If $C^I_{\vec{\alpha_2}} \vdash l \leq (l \sqcup S)$ by [LubR] for some set S, there is nothing to prove, since by [LubR] we have $C^I_{\vec{\alpha_1}} \vdash \phi(l) \leq (\phi(l) \sqcup \phi(S))$. Otherwise, we have $C^I_{\vec{\alpha_2}} \vdash l \leq l''$ for some $l' \in \vec{\alpha_2}$. Then l'' has polarity $-$ in τ_2, and thus $l'' \leq \phi(l'')$. Then $C^I \vdash \phi(l) \leq \phi(l'')$, and thus $C^I_{\vec{\alpha_1}} \vdash \phi(l) \leq \phi(l'')$, since $\phi(l), \phi(l'') \in \vec{\alpha_1}$.

Thus we have $C^I_{\vec{\alpha_1}} \vdash \phi(C^I_{\vec{\alpha_2}})$. Notice that there is not requirement that these constraints are part of C^I_T, which follows the COPY system pattern that constraints on existential types do not "leak" out to the outer constraint context upon subtyping them.

Finally, by alpha-conversion we have $\exists^{l_2}\vec{\alpha_2}[C^I_{\vec{\alpha_2}}].\tau_2 = \exists^{l_2}\phi(\vec{\alpha_2})[\phi(C^I_{\vec{\alpha_2}})].\phi(\tau_2)$

Thus we have

$$[\text{Sub-}\exists]\frac{\begin{array}{c} C^I_{\vec{\alpha_1}} \vdash \phi(C^I_{\vec{\alpha_2}}) \\ D_1 = (\Psi_{C,I}(D))[l \mapsto (\Psi_{C,I}(D))(l) + 1, \forall l \in \vec{\alpha_1}] \\ C^I_T; D_1 \vdash \Psi_{C,I}(\tau_1) \leq \phi(\Psi_{C,I}(\tau_2)) \\ C^I_T; \Psi_{C,I}(D) \vdash l_1 \leq l_2 \end{array}}{C^I_T; \Psi_{C,I}(D) \vdash \exists^{l_1}\vec{\alpha_1}[C^I_{\vec{\alpha_1}}].\Psi_{C,I}(\tau_1) \leq \exists^{l_2}\vec{\alpha_2}[C^I_{\vec{\alpha_2}}].\Psi_{C,I}(\tau_2)}$$

$\square$

Theorem 37 (Reduction from CFL to COPY) *Let $\mathcal{D}$ be a normal CFL derivation of $I; C; \Gamma \vdash_{CFL} e : \tau$. Then $C^I_{(fl(\Gamma) \cup fl(\tau))}; \Psi_{C,I}(\Gamma) \vdash_{cp} e : \Psi_{C,I}(\tau)$.*

Proof: By induction on the given CFL derivation. As a shorthand notation in the proof, we define C^I_Γ as $C^I_{fl(\Gamma)}$, C^I_τ as $C^I_{fl(\tau)}$, and we use commas in place of unions when subscripting.

Case [Id]. We have

$$[\text{Id (CFL)}]\frac{}{I; C; \Gamma, x : \tau \vdash_{CFL} x : \tau}$$

Thus trivially

$$[\text{Id (COPY)}]\frac{}{C^I_{(\Gamma,\tau)}; \Psi_{C,I}(\Gamma), x : \Psi_{C,I}(\tau) \vdash_{cp} x : \Psi_{C,I}(\tau)}$$

Case [Int]. We have

$$[\text{Int (CFL)}]\frac{C \vdash L \leq l}{I; C; \Gamma \vdash_{CFL} n^L : int^l}$$

Then since $C \vdash L \leq l$ and $l \in fl(int^l) = \{l\}$ we have $C^I_{(\Gamma,l)} \vdash L \leq l$. Thus

$$[\text{Int (COPY)}]\frac{C^I_{(\Gamma,l)} \vdash L \leq l}{C^I_{(\Gamma,l)}; \Psi_{C,I}(\Gamma) \vdash_{cp} n^L : int^l}$$

and $\Psi_{C,I}(int^l) = int^l$.

Case [Lam]. We have

$$[\text{Lam (CFL)}]\frac{I; C; \Gamma, x : \tau \vdash_{CFL} e : \tau' \qquad C \vdash L \leq l}{I; C; \Gamma \vdash_{CFL} \lambda^L x.e : \tau \rightarrow^l \tau'}$$

Then since $l \in fl(\tau \rightarrow^l \tau')$ we have $C^I_{(\Gamma,\tau,\tau',l)} \vdash L \leq l$. By induction, $C^I_{(\Gamma,\tau,\tau')}; \Psi_{C,I}(\Gamma), x : \Psi_{C,I}(\tau) \vdash_{cp} e : \Psi_{C,I}(\tau')$. Then by Lemmas 22 and 6, we have $C^I_{(\Gamma,\tau,\tau',l)}; \Psi_{C,I}(\Gamma), x : \Psi_{C,I}(\tau) \vdash_{cp} e :$

$\Psi_{C,I}(\tau')$. Thus we have

$$[\text{Lam (Copy)}]\frac{C^I_{(\Gamma,\tau,\tau',l)}; \Psi_{C,I}(\Gamma), x : \Psi_{C,I}(\tau) \vdash_{cp} e : \Psi_{C,I}(\tau') \qquad C^I_{(\Gamma,\tau,\tau',l)} \vdash L \leq l}{C^I_{(\Gamma,\tau,\tau',l)}; \Psi_{C,I}(\Gamma) \vdash_{cp} \lambda^L x.e : \Psi_{C,I}(\tau) \to^l \Psi_{C,I}(\tau')}$$

and $\Psi_{C,I}(\tau \to^l \tau') = \Psi_{C,I}(\tau) \to^l \Psi_{C,I}(\tau')$.

Case [App]. We have

$$[\text{App (CFL)}]\frac{\begin{array}{c} I; C; \Gamma \vdash_{CFL} e_1 : \tau \to^l \tau' \\ I; C; \Gamma \vdash_{CFL} e_2 : \tau \\ C \vdash l \leq L \end{array}}{I; C; \Gamma \vdash_{CFL} e_1 @^L e_2 : \tau'}$$

Let $\psi = \psi_{(\Gamma,\tau')}$. By induction, $C^I_{(\Gamma,\tau,\tau',l)}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_1 : \Psi_{C,I}(\tau) \to^l \Psi_{C,I}(\tau')$. Then

$$\psi(C^I_{(\Gamma,\tau,\tau',l)}); \psi(\Psi_{C,I}(\Gamma)) \vdash_{cp} e_1 : \psi(\Psi_{C,I}(\tau) \to^l \Psi_{C,I}(\tau'))$$

But $\psi(C^I_{(\Gamma,\tau,\tau',l)}) = C^I_{(\Gamma,\tau')}$ and $\psi(\Psi_{C,I}(\Gamma)) = \Psi_{C,I}(\Gamma)$ by Lemma 30. Similarly, $\psi(\Psi_{C,I}(\tau')) = \Psi_{C,I}(\tau')$. Thus

$$C^I_{(\Gamma,\tau')}, \Psi_{C,I}(\Gamma) \vdash_{cp} e_1 : \psi(\Psi_{C,I}(\tau)) \to^{\psi(l)} \Psi_{C,I}(\tau')$$

Also by induction, $C^I_{(\Gamma,\tau)}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_2 : \Psi_{C,I}(\tau)$, and by similar reasoning and Lemma 22 we get $C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_2 : \psi(\Psi_{C,I}(\tau))$.

Finally, since $C \vdash l \leq L$, we have $C^I_{(\Gamma,\tau')} \vdash \psi(l) \leq \psi(L)$ or $C^I_{(\Gamma,\tau')} \vdash \psi(l) \leq L$.

But then we have

$$[\text{App (Copy)}]\frac{\begin{array}{c} C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_1 : \psi(\Psi_{C,I}(\tau)) \to^{\psi(l)} \Psi_{C,I}(\tau') \\ C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_2 : \psi(\Psi_{C,I}(\tau)) \qquad C^I_{(\Gamma,\tau')} \vdash \psi(l) \leq L \end{array}}{C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_1 @^L e_2 : \Psi_{C,I}(\tau')}$$

Case [Pair], [Proj], [Cond]. Similar to [App].

Case [Sub]. We have

$$[\text{Sub (CFL)}]\frac{I; C; \Gamma \vdash_{CFL} e : \tau \qquad C; \emptyset \vdash \tau \leq \tau'}{I; C; \Gamma \vdash_{CFL} e : \tau'}$$

By induction and Lemma 22, we have $C^I_{(\Gamma,\tau,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e : \Psi_{C,I}(\tau)$. Let $\psi = \psi_{(\Gamma,\tau')}$. Then

$$\psi(C^I_{(\Gamma,\tau,\tau')}); \psi(\Psi_{C,I}(\Gamma)) \vdash_{cp} e : \psi(\Psi_{C,I}(\tau))$$

But by Lemma 30 we have $\psi(\Psi_{C,I}(\Gamma)) = \Psi_{C,I}(\Gamma)$. And $\psi(C^I_{(\Gamma,\tau,\tau')}) = C^I_{(\Gamma,\tau')}$. Thus we have

$$C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e : \psi(\Psi_{C,I}(\tau))$$

Next, by Lemma 36, we have $C^I_{(\tau,\tau')} \vdash_{cp} \Psi_{C,I}(\tau) \leq \Psi_{C,I}(\tau')$. Thus by Lemma 22 we have $C^I_{(\Gamma,\tau,\tau')} \vdash_{cp} \Psi_{C,I}(\tau) \leq \Psi_{C,I}(\tau')$ Then

$$\psi(C^I_{(\Gamma,\tau,\tau')}) \vdash_{cp} \psi(\Psi_{C,I}(\tau)) \leq \psi(\Psi_{C,I}(\tau'))$$

But $\psi(\Psi_{C,I}(\tau')) = \Psi_{C,I}(\psi(\tau'))$ by Lemma 29, and $\Psi_{C,I}(\psi(\tau')) = \Psi_{C,I}(\tau')$ by definition of ψ. And $\psi(C^I_{(\Gamma,\tau,\tau')}) = C^I_{(\Gamma,\tau')}$. Thus by Lemma 22 we have

$$C^I_{(\Gamma,\tau')} \vdash_{cp} \psi(\Psi_{C,I}(\tau)) \leq \Psi_{C,I}(\tau')$$

Then we have

$$[\text{Sub (Copy)}]\frac{\begin{array}{c} C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e : \psi(\Psi_{C,I}(\tau)) \\ C^I_{(\Gamma,\tau')}; \emptyset \vdash_{cp} \psi(\Psi_{C,I}(\tau)) \leq \Psi_{C,I}(\tau') \end{array}}{C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e : \Psi_{C,I}(\tau')}$$

Case [Let]. We have

$$[\text{Let (CFL)}]\frac{\begin{array}{cc} I; C; \Gamma \vdash_{CFL} e_1 : \tau_1 & I; C; \Gamma, f : (\forall \vec{\alpha}.\tau_1, \vec{l}) \vdash_{CFL} e_2 : \tau_2 \\ \vec{\alpha} = fl(\tau_1) \setminus \vec{l} & \vec{l} = fl(\Gamma) \end{array}}{I; C; \Gamma \vdash_{CFL} \texttt{let}\ f = e_1\ \texttt{in}\ e_2 : \tau_2}$$

By induction, we have $C^I_{(\Gamma,\tau_1)}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_1 : \Psi_{C,I}(\tau_1)$. But $\vec{l} = fl(\Gamma)$, and $\vec{\alpha} = fl(\tau_1) \setminus \vec{l}$. Thus $fl(\Gamma) \cup fl(\tau_1) = \vec{\alpha} \cup \vec{l}$. Therefore $C^I_{(\vec{\alpha},\vec{l})}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_1 : \Psi_{C,I}(\tau_1)$.

Then since $\vec{l} = fl(\Gamma)$, by induction we also have $C^I_{(\Gamma,\tau_2)}; \Psi_{C,I}(\Gamma_{cp}), f : \forall \vec{\alpha}[C^I_{(\vec{\alpha},\vec{l})}].(\Psi_{C,I}(\tau_1)) \vdash_{cp} e_2 : \Psi_{C,I}(\tau_2)$.

Finally, by Lemma 31 we have $fl(\Psi_{C,I}(\tau_1)) = fl(\tau_1)$, and by Lemma 32 we have $fl(\Psi_{C,I}(\Gamma)) \subseteq fl(\Gamma)$. Thus

$$\vec{\alpha} = fl(\tau_1) \setminus \vec{l} \subseteq \left(fl(\Psi_{C,I}(\tau_1)) \cup fl(C^I_{(\vec{\alpha},\vec{l}))}) \right) \setminus fl(\Psi_{C,I}(\Gamma))$$

Therefore we can apply [Let] rule of the Copy system to prove

$$[\text{Let (Copy)}]\frac{\begin{array}{c} C^I_{(\vec{\alpha},\vec{l})}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_1 : \Psi_{C,I}(\tau_1) \\ C^I_{(\Gamma,\tau_2)}; \Psi_{C,I}(\Gamma_{cp}), f : \forall \vec{\alpha}[C^I_{(\vec{\alpha},\vec{l})}].(\Psi_{C,I}(\tau_1)) \vdash_{cp} e_2 : \Psi_{C,I}(\tau_2) \\ \vec{\alpha} \subseteq \left(fl(\Psi_{C,I}(\tau_1)) \cup fl(C^I_{(\vec{\alpha},\vec{l}))}) \right) \setminus fl(\Psi_{C,I}(\Gamma)) \end{array}}{C^I_{(\Gamma,\tau_2)}; \Psi_{C,I}(\Gamma) \vdash_{cp} \texttt{let}\ f = e_1\ \texttt{in}\ e_2 : \Psi_{C,I}(\tau_2)}$$

Case [Fix]. Similar to [Let] and [Inst]

Case [Inst]. We have

$$[\text{Inst (CFL)}]\frac{\begin{array}{c} I; C; \emptyset \vdash \tau \preceq^i_+ \tau' : \phi \\ dom(\phi) = \vec{\alpha} \quad I \vdash \vec{l} \preceq^i_+ \vec{l} \quad I \vdash \vec{l} \preceq^i_- \vec{l} \end{array}}{I; C; \Gamma, f : (\forall \vec{\alpha}.\tau, \vec{l}) \vdash_{CFL} f^i : \tau'}$$

By definition $\Psi_{C,I}((\forall\vec{\alpha}.\tau, \vec{l})) = \forall\vec{\alpha}[C^I_{(\vec{\alpha},\vec{l})}].(\Psi_{C,I}(\tau))$. Notice that $\vec{l}$ is the set of free labels at the point where f was bound by [Let] or [Fix], and that this use of [Inst] is nested inside that derivation. Thus $\vec{l} \subseteq fl(\Gamma)$, and by [Inst (CFL)] we have $\phi(\vec{l}) = \vec{l}$. Further, since $\phi(\tau) = \tau'$ we have $\phi(\vec{\alpha}) \subseteq fl(\tau')$. Thus $\phi(\vec{\alpha} \cup \vec{l}) \subseteq fl(\tau') \cup fl(\Gamma)$. Then by Lemma 25, we have $C^I_{(\Gamma,\tau')} \vdash \hat{\phi}(C^I_{(\vec{\alpha},\vec{l})})$.

Thus we can apply the [Inst] rule of Copy:

$$\text{[Inst (Copy)]}\frac{C^I_{(\Gamma,\tau')} \vdash \hat{\phi}(C^I_{(\vec{\alpha},\vec{l})})}{C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma), f : \forall\vec{\alpha}[C^I_{(\vec{\alpha},\vec{l})}].(\Psi_{C,I}(\tau)) \vdash_{cp} f^i : \hat{\phi}(\Psi_{C,I}(\tau))}$$

Finally, by Lemma 29 we have $\hat{\phi}(\Psi_{C,I}(\tau)) = \Psi_{C,I}(\hat{\phi}(\tau)) = \Psi_{C,I}(\tau')$,

Case [Pack]. We have

$$\text{[Pack (CFL)]}\frac{\begin{array}{cc} I; C; \Gamma \vdash_{CFL} e : \tau' & I; C; \emptyset \vdash \tau \preceq^i_- \tau' : \phi \\ dom(\phi) = \vec{\alpha} & C \vdash L \leq l \end{array}}{I; C; \Gamma \vdash_{CFL} \mathtt{pack}^{L,i}\ e : \exists^l\vec{\alpha}.\tau}$$

Since $C \vdash L \leq l$ we have $C^I_{(\Gamma,l,\tau')} \vdash L \leq l$. By definition $\Psi_{C,I}(\exists^l\vec{\alpha}.\tau) = \exists^l\vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau))$. By induction and Lemma 22 we have $C^I_{(\Gamma,l,\tau')}, \Psi_{C,I}(\Gamma) \vdash e : \Psi_{C,I}(\tau')$. But $\phi(\tau) = \tau'$, so by Lemma 29 we have $\Psi_{C,I}(\tau') = \check{\phi}(\Psi_{C,I}(\tau))$. Also, since the instantiation context is normal, and since $\phi(\vec{\alpha}) \subseteq fl(\tau')$, by Lemma 34 we have $C^I_{(\Gamma,l,\tau')} \vdash \check{\phi}(C^I_{\vec{\alpha}})$. Putting this together yields

$$\text{[Pack (Copy)]}\frac{\begin{array}{cc} C^I_{(\Gamma,l,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e : \check{\phi}(\Psi_{C,I}(\tau)) & C^I_{(\Gamma,l,\tau')} \vdash \check{\phi}(C^I_{\vec{\alpha}}) \\ \multicolumn{2}{c}{C^I_{(\Gamma,l,\tau')} \vdash L \leq l} \end{array}}{C^I_{(\Gamma,l,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} \mathtt{pack}^{L,i}\ e : \exists^l\vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau))}$$

Let $\psi = \psi_{(\Gamma,l,(\tau\setminus\vec{\alpha}))}$. Then we have

$$\psi(C^I_{(\Gamma,l,\tau')}); \psi(\Psi_{C,I}(\Gamma)) \vdash_{cp} \mathtt{pack}^{L,i}\ e : \psi(\exists^l\vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau)))$$

Notice that $fl(\tau) \setminus \vec{\alpha} \subseteq fl(\tau')$. Then $\psi(C^I_{(\Gamma,l,\tau')}) = C^I_{(\Gamma,l,(\tau\setminus\vec{\alpha}))}$. And by Lemma 30 we have $\psi(\Psi_{C,I}(\Gamma)) = \Psi_{C,I}(\Gamma)$. Finally, $\psi(\exists^l\vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau))) = \exists^l\vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau))$, since $\psi(l) = l$ by definition, all the labels in $C^I_{\vec{\alpha}}$ are bound, and since $fl(\Psi_{C,I}(\tau)) = fl(\tau)$ by Lemma 31 and the only unbound labels of τ are those in $fl(\tau) \setminus \vec{\alpha}$, which ψ does not affect by definition. Thus

$$C^I_{(\Gamma,l,(\tau\setminus\vec{\alpha}))}; \Psi_{C,I}(\Gamma) \vdash_{cp} \mathtt{pack}^{L,i}\ e : \exists^l\vec{\alpha}[C^I_{\vec{\alpha}}].(\Psi_{C,I}(\tau))$$

Case [Unpack]. We have

$$\text{[Unpack (CFL)]}\frac{\begin{array}{c} I; C; \Gamma \vdash_{CFL} e_1 : \exists^l\vec{\alpha}.\tau \qquad I; C; \Gamma, x : \tau \vdash_{CFL} e_2 : \tau' \\ \vec{l} = fl(\Gamma) \cup (fl(\tau) \setminus \vec{\alpha}) \cup fl(\tau') \cup L \qquad \vec{\alpha} \subseteq fl(\tau) \setminus \vec{l} \qquad C \vdash l \leq L \\ \forall l \in \vec{\alpha}, l' \in \vec{l}.(I; C \not\vdash l \rightsquigarrow_m l' \text{ and } I; C \not\vdash l' \rightsquigarrow_m l) \end{array}}{I; C; \Gamma \vdash_{CFL} \mathtt{unpack}^L\ x = e_1\ \mathtt{in}\ e_2 : \tau'}$$

By induction and Lemma 5, we have $C^I_{(\Gamma,l,(\tau\setminus\vec{\alpha}),\tau')}; \Psi_{C,I}(\Gamma) \vdash e_1 : \exists^l \vec{\alpha}[C^I_{\vec{\alpha}}].\Psi_{C,I}(\tau)$ Let $\psi = \psi_{(\Gamma,\tau')}$. Then we have

$$\psi(C^I_{(\Gamma,l,(\tau\setminus\vec{\alpha}),\tau')}); \psi(\Psi_{C,I}(\Gamma)) \vdash e_1 : \psi(\exists^l \vec{\alpha}[C^I_{\vec{\alpha}}].\Psi_{C,I}(\tau))$$

Then by Lemma 30 we have $\psi(\Psi_{C,I}(\Gamma)) = \Psi_{C,I}(\Gamma)$. Also, we have $\psi(C^I_{(\Gamma,l,(\tau\setminus\vec{\alpha}))}) = C^I_{(\Gamma,\tau')}$. Finally, notice that $\psi(\exists^l \vec{\alpha}[C^I_{\vec{\alpha}}].\Psi_{C,I}(\tau)) = \exists^{\psi(l)} \vec{\alpha}[C^I_{\vec{\alpha}}].\psi'(\Psi_{C,I}(\tau))$, where $\psi'(l) = l$ if $l \in \vec{\alpha}$ and $\psi'(l) = \psi(l)$ otherwise. Further, all labels in $C^I_{\vec{\alpha}}$ are bound. Thus we have

$$C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash e_1 : \exists^{\psi(l)} \vec{\alpha}[C^I_{\vec{\alpha}}].\psi'(\Psi_{C,I}(\tau))$$

Also by induction $C^I_{(\Gamma,\tau,\tau')}; \Psi_{C,I}(\Gamma), x : \Psi_{C,I}(\tau) \vdash e_2 : \Psi_{C,I}(\tau')$. Then we claim

$$C^I_{(\Gamma,(\tau\setminus\vec{\alpha}),\tau')} \cup C^I_{\vec{\alpha}} \vdash C^I_{(\Gamma,(\tau\setminus\vec{\alpha}),\vec{\alpha},\tau')} = C^I_{(\Gamma,\tau,\tau')}$$

To see why, suppose $C^I_{(\Gamma,\tau,(\tau\setminus\vec{\alpha}),\tau')} \vdash l \leq l'$. Then without loss of generality, assume l and l' are labels rather than joins. If l or l' is in L, then the result holds trivially. Also, if $l, l' \in fl(\Gamma) \cup (fl(\tau)\setminus\vec{\alpha}) \cup fl(\tau')$ or $l, l' \in \vec{\alpha}$, then the result holds trivially. Otherwise, we have $I; C \vdash l \leadsto_m l'$ with one of l, l' in $fl(\Gamma) \cup (fl(\tau) \setminus \vec{\alpha}) \cup fl(\tau')$ and one in $\vec{\alpha}$, which is impossible by the last hypothesis of [Unpack]. Thus by Lemma 5 we have

$$C^I_{(\Gamma,(\tau\setminus\vec{\alpha}),\tau')} \cup C^I_{\vec{\alpha}}; \Psi_{C,I}(\Gamma), x : \Psi_{C,I}(\tau) \vdash e_2 : \Psi_{C,I}(\tau')$$

But then we have

$$\psi'(C^I_{(\Gamma,(\tau\setminus\vec{\alpha}),\tau')} \cup C^I_{\vec{\alpha}}); \psi'(\Psi_{C,I}(\Gamma)), x : \psi'(\Psi_{C,I}(\tau)) \vdash e_2 : \psi'(\Psi_{C,I}(\tau'))$$

ψ' and ψ only differ on $\vec{\alpha}$, and by assumption $\vec{\alpha} \cap \vec{l} = \emptyset$. Thus by Lemma 30 we have $\psi'(\Psi_{C,I}(\Gamma)) = \Gamma$ and $\psi'(\Psi_{C,I}(\tau')) = \Psi_{C,I}(\tau')$. Further, ψ' does not affect $\vec{\alpha}$, so $\psi'(C^I_{(\Gamma,(\tau\setminus\vec{\alpha}),\tau')} \cup C^I_{\vec{\alpha}}) = \psi'(C^I_{(\Gamma,(\tau\setminus\vec{\alpha}),\tau')}) \cup C^I_{\vec{\alpha}}$. And $\psi'(C^I_{(\Gamma,(\tau\setminus\vec{\alpha}),\tau')}) = C^I_{(\Gamma,\tau')}$, again since $\vec{\alpha} \cap \vec{l} = \emptyset$. Putting this all together, we have

$$C^I_{(\Gamma,\tau')} \cup C^I_{\vec{\alpha}}; \Psi_{C,I}(\Gamma), x : \psi'(\Psi_{C,I}(\tau)) \vdash e_2 : \Psi_{C,I}(\tau')$$

Finally, since $C \vdash l \leq L$, we have $C^I_{(\Gamma,\tau')} \vdash \psi(l) \leq \psi(L)$ or $C^I_{(\Gamma,\tau')} \vdash \psi(l) \leq L$. Also, $\vec{\alpha} \subseteq fl(\tau) \setminus \vec{l}$. By Lemma 32 we have $fl(\Psi_{C,I}(\Gamma)) \subseteq fl(\Gamma)$. By Lemma 31 we have $fl(\Psi_{C,I}(\tau')) = fl(\tau')$. And $fl(C^I_{(\Gamma,\tau')}) \subseteq fl(\Gamma) \cup fl(\tau')$. And since $\vec{l} \supseteq fl(\Gamma) \cup fl(\tau')$ we have

$$\vec{\alpha} \subseteq \big(fl(\Psi_{C,I}(\tau)) \cup fl(C^I_{\vec{\alpha}})\big) \setminus \Big(fl(\Psi_{C,I}(\Gamma)) \cup fl(C^I_{(\Gamma,\tau')}) \cup fl(\Psi_{C,I}(\tau'))\Big)$$

Putting these all together, we get

$$\text{[Unpack (Copy)]}\ \frac{\begin{array}{c} C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} e_1 : \exists^{\psi(l)} \vec{\alpha}[C^I_{\vec{\alpha}}].\psi'(\Psi_{C,I}(\tau)) \qquad C^I_{(\Gamma,\tau')} \vdash \psi(l) \leq L \\ C^I_{(\Gamma,\tau')} \cup C^I_{\vec{\alpha}}; \Psi_{C,I}(\Gamma), x : \psi'(\Psi_{C,I}(\tau)) \vdash_{cp} e_2 : \Psi_{C,I}(\tau') \\ \vec{\alpha} \subseteq \big(fl(\Psi_{C,I}(\tau)) \cup fl(C^I_{\vec{\alpha}})\big) \setminus \Big(fl(\Psi_{C,I}(\Gamma)) \cup fl(C^I_{(\Gamma,\tau')}) \cup fl(\Psi_{C,I}(\tau'))\Big) \end{array}}{C^I_{(\Gamma,\tau')}; \Psi_{C,I}(\Gamma) \vdash_{cp} \texttt{unpack}^L\ x = e_1\ \texttt{in}\ e_2 : \Psi_{C,I}(\tau')}$$

$\square$