Faithful Translations between Polyvariant Flows and Polymorphic Types

Torben Amtoft*
Department of Computer Science
Boston University
Boston, MA 02215, U.S.A.
tamtoft@cs.bu.edu
http://www.cs.bu.edu/associates/~tamtoft

Franklyn A. Turbak†
Computer Science Department
Wellesley College
Wellesley, MA 02482, U.S.A.
fturbak@wellesley.edu
http://cs.wellesley.edu/~fturbak

Boston University Computer Science Department

Draft of Technical Report BUCS-TR-2001-XX

July 31, 2001

This document is a draft of a technical report that expands upon [AT00] and is in preparation for submission to ACM Transactions on Programming Languages and Systems.

*Supported by NSF grant EIA–9806745.
†Supported by NSF grant EIA–9806747.

Abstract

Recent work has shown equivalences between various type systems and flow logics. Ideally, the translations upon which such equivalences are based should be *faithful* in the sense that information is not lost in round-trip translations from flows to types and back or from types to flows and back. Building on the work of Nielson & Nielson and of Palsberg & Pavlopoulou, we present the first faithful translations between a class of finitary polyvariant flow analyses and a type system supporting polymorphism in the form of intersection and union types. Additionally, our flow/type correspondence solves several open problems posed by Palsberg & Pavlopoulou: (1) it expresses call-string based polyvariance (such as k-CFA) as well as argument based polyvariance; (2) it enjoys a subject reduction property for flows as well as for types; and (3) it supports a flow-oriented perspective rather than a type-oriented one.

1 Introduction

Type systems and flow logic are two popular frameworks for specifying program analyses. While these frameworks seem rather different on the surface, both describe the "plumbing" of a program, and recent work has uncovered deep connections between them. For example, Palsberg and O'Keefe [PO95] demonstrated an equivalence between determining flow safety in the monovariant 0-CFA flow analysis and typability in a system with recursive types and subtyping [AC93]. Heintze showed equivalences between four restrictions of 0-CFA and four type systems parameterized by (1) subtyping and (2) recursive types [Hei95].

Because they merge flow information for all calls to a function, monovariant analyses are imprecise. Greater precision can be obtained via polyvariant analyses, in which functions can be analyzed in multiple abstract contexts. Examples of polyvariant analyses include call-string based approaches, such as k-CFA [Shi91, JW95, NN97], polymorphic splitting [WJ98], type-directed flow analysis [JWW97], and argument based polyvariance, such as Schmidt's analysis [Sch95] and Agesen's cartesian product analysis [Age95]. In terms of the flow/type correspondence, several forms of flow polyvariance appear to correspond to type polymorphism expressed with intersection and union types [Ban97, WDMT97a, DMTW97, PP01]. Intuitively, intersection types are finitary polymorphic types that model the multiple analyses for a given abstract closure, while union types are finitary existential types that model the merging of abstract values where flow paths join. Palsberg and Pavlopoulou (henceforth P&P) were the first to formalize this correspondence by demonstrating an equivalence between a class of flow analyses supporting argument based polyvariance and a type system with union and intersection types [PP01].

If type and flow systems encode similar information, translations between the two should be *faithful*, in the sense that round-trip translations from flow analyses to type derivations and back (or from type derivations to flow analyses and back) should not lose precision. Faithfulness formalizes the intuitive notion that a flow analysis and its corresponding type derivation contain the same information content. Interestingly, neither the translations of Palsberg and O'Keefe nor those of P&P are faithful. The lack of faithfulness in P&P is demonstrated by a simple example. Let $e = \big(\lambda^1\mathtt{x}.\mathtt{succ\ x}\big) @ \big(\big(\lambda^2\mathtt{y}.\mathtt{y}\big) @ 3\big)$, where we have labeled two program points of interest. Consider an initial monovariant flow analysis in which the only abstract closure reaching point 1 is $v_1 = (\lambda\mathtt{x}.\mathtt{succ\ x}, \epsilon)$ and the only one reaching point 2 is $v_2 = (\lambda\mathtt{y}.\mathtt{y}, \epsilon)$. The flow-to-type translation of P&P yields the expected type derivation:

$$\dfrac{\dfrac{\cdots}{\epsilon \vdash \lambda^1\mathtt{x}.\mathtt{succ\ x} : \mathtt{int} \to \mathtt{int}} \qquad \dfrac{\dfrac{\cdots}{\epsilon \vdash \lambda^2\mathtt{y}.\mathtt{y} : \mathtt{int} \to \mathtt{int}} \quad \cdots}{\epsilon \vdash (\lambda^2\mathtt{y}.\mathtt{y}) @ 3 : \mathtt{int}}}{\epsilon \vdash (\lambda^1\mathtt{x}.\mathtt{succ\ x}) @ ((\lambda^2\mathtt{y}.\mathtt{y}) @ 3) : \mathtt{int}}$$

However, P&P's type-to-flow translation loses precision by merging into a single set all abstract closures associated with the same type in a given derivation. For the example derivation above, the type $\mathtt{int} \to \mathtt{int}$ translates back to the abstract closure set $V = \{v_1, v_2\}$, yielding a less precise flow analysis in which V flows to both points 1 and 2.

In contrast, Heintze's translations are faithful. The undesirable merging in the above example is avoided by annotating function types with a label set indicating the source point of the function value. Thus, $\lambda^1\mathtt{x}.\mathtt{succ\ x}$ has type $\mathtt{int} \overset{\{1\}}{\to} \mathtt{int}$ while $\lambda^2\mathtt{y}.\mathtt{y}$ has type $\mathtt{int} \overset{\{2\}}{\to} \mathtt{int}$.

1.1 Contributions of this Paper

In this paper, we present the first faithful translations between a broad class of polyvariant flow analyses and a type system with polymorphism in the form of intersection and union types. The translations are faithful in the sense that a round-trip translation acts as the identity for canonical types/flows, and otherwise canonicalizes. In particular, our round-trip translation for types preserves non-recursive types that P&P may transform to recursive types. We achieve this result by adapting the translations of P&P to use a modified version of the flow analysis framework of Nielson and Nielson (henceforth N&N) [NN97]. As in Heintze's translations, annotations play a key role in the faithfulness of our translations: we (1) annotate flow values to indicate the sinks to which they flow, and (2) annotate union and intersection types with component labels that serve as witnesses for existential quantifiers that appear in the definition of subtyping. These annotations can be justified purely in terms of the type or flow system, independent of the flow/type correspondence.

Additionally, our framework solves several open problems posed by P&P:

1. *Unifying P&P and N&N*: Whereas P&P's flow specification can readily handle only argument based polyvariance, N&N's flow specification can also express call-string based polyvariance. So our translations give the first type system corresponding to k-CFA analysis where $k \geq 1$.

2. *Subject reduction for flows*: We inherit from N&N's flow logic the property that flow information valid before a reduction step is still valid afterwards. In contrast, P&P's flow system does not have this property. (Both our system and P&P have subject reduction for types.)

3. *Letting "flows have their way"*: P&P discuss mismatches between flow and type systems that imply the need to choose one perspective over the other when designing a translation between the two systems. In their translations, P&P choose to always let types "have their way"; for example they require analyses to be finitary and to analyze all closure bodies, even though they may be dead code. In contrast, our design also lets flows "have their way", in that our type system does not require all subexpressions to be analyzed.

1.2 Motivation

While the relationship between flow logics and type systems is an intriguing theoretical question, it has important practical ramifications as well. Flow information is useful for guiding and/or enhancing a wide variety of analyses and optimizations, such as closure conversion([SW97, DWM+01]), defunctionalization([Tol97, TO98, CJW00], inlining([WJ98]), uncurrying ([HH98]). eager thunk evaluation ([Fax93]), dead code elimination ([WS99]), run-time check elimination ([WJ98]), loop detection[SGL96], and object specialization ([DCG95, PC95]). Encoding flow information into type systems enables type-directed compilers to support such flow-directed optimizations in a uniform rather than ad hoc fashion, with all the usual attendant benefits of using a typed intermediate language (e.g.,[TMC+96, PJ96, MWCG99]). For instance, flow information can be preserved by one compiler transformation so that it is available for subsequent passes, and the additional flow information aids in debugging the implementation of the transformations [DMTW97]. In this context, better understanding of the relationship between flows and types can lead to improvements in state-of-the-art compiler technology. Indeed, our motivation for this work is to formalize the encoding of flow information in the intersection and union types of CIL, the intermediate language used in the Church Project[1] compiler [WDMT97a].

1.3 Overview of Paper

Sect. 2 presents the source language. Our type system is introduced in Sect. 3 and our flow framework in Sect. 4. Sects. 5 and 6 present the type-to-flow and flow-to-type translations,

[1]The work reported here is part of the Church Project (`http://www.cs.bu.edu/groups/church/`), whose goal is to study sophisticated type systems and their application to programming language design and implementation.

e	$\in$	**LabExpr** $::= ue^l$	(labeled expressions)
l	$\in$	**Label**	(infinite set of labels)
ue	$\in$	**UnLabExpr** $::=$	(unlabeled expressions)
		$z \mid fn \mid e\,@\,e \mid c \mid \texttt{succ}\ e \mid \texttt{if0}\ e\ \texttt{then}\ e\ \texttt{else}\ e$	(pure expressions)
		$\mid bd \mid cl$	(impure expressions)
z	$\in$	**Var** $::= f \mid x$	(infinite set of variables)
f	$\in$	**MuVar**	(μ-bound variables)
x	$\in$	**LamVar**	(λ-bound variables)
fn	$\in$	**Abstr** $::= \mu f.\lambda x.e$	(abstractions)
c	$\in$	**Int** $::= \{-2, -1, 0, 1, 2, \ldots\}$	(integers)
bd	$\in$	**Binder** $::= \texttt{bind}\ se\ \texttt{in}\ e$	(binders)
cl	$\in$	**Closure** $::= \texttt{close}\ fn\ \texttt{in}\ se$	(closures)
se	$\in$	**SemEnv** $::= \epsilon \mid se[z \mapsto sv]$	(semantic environments)
sv	$\in$	**Value** $::= c \mid cl$	(semantic values)

Figure 1: Syntax of the language $\mathfrak{L}$.

respectively, while round-trip translations are discussed in Sect. 7. Sect. 8 concludes with a discussion of future work.

2 The Language

2.1 Syntax

We study the relationship between types and flows in the context of an extended λ-calculus we call $\mathfrak{L}$. The syntax of $\mathfrak{L}$ is presented in Fig. 1. The metavariable *fn* ranges over unlabeled function abstractions of the form $\mu f.\lambda x.e$, which denotes a function with parameter x which may call itself via f. We use the notation $\lambda x.e$ as a shorthand for $\mu f.\lambda x.e$ where f does not occur in e.[2] μ-bound variables (f) and λ-bound variables (x) are distinct; z ranges over both. In examples, we use the concrete variables `x`, `y` and `z` for λ-bound variables; `g` is a λ-bound variable assumed to be bound to a function. Function applications use the explicit application symbol @, which serves as a convenient location for labels (see below). $\mathfrak{L}$ also supports integer constants (c), the successor function (`succ`), and the ability to test for zero (`if0`). It would be straightforward to add other constructs (e.g., other constants and operations, products, sums, binding constructs[3]), but to keep the exposition simple we shall refrain from doing so.

As in N&N, we use `bind` (*bd*) and `close` (*cl*) expressions to represent "intermediate configurations" in the semantics of $\mathfrak{L}$ (see Sec. 2.2); these should not appear in initial "user" programs. These configurations are defined in terms of *semantic environments* (*se*) that map variables to a distinguished subset of *unlabeled* expressions called *values*. A value (*sv*) is either a number (c) or a *closure* of the form $\texttt{close}\ fn\ \texttt{in}\ se$. Semantic environments are lists of bindings of the form $[z \mapsto sv]$. The empty environment is denoted ϵ, and the juxtaposition notation $se[z \mapsto sv]$ denotes the environment that results from extending se with the binding $[z \mapsto sv]$. We write $\mathcal{E}[z_1, z_2 \mapsto sv]$ for $\mathcal{E}[z_1 \mapsto sv][z_2 \mapsto sv]$. We define $dom(se)$ as $\{z \mid [z \mapsto sv] \in se\}$ and $cod(se)$ as $\{sv \mid [z \mapsto sv] \in se\}$. We also treat an

[2] Our formal development assumes that all abstractions are of the form $\mu f.\lambda x.e$, but in the text we often treat abstractions of the form $\lambda x.e$ specially in order to reduce clutter. In particular, we do not show "useless" bindings involving an unreferenced μ-bound variable f even though these are technically required by the formalism. The reader may imagine that we employ an extended version of our formal system where $\lambda x.e$ is explicitly handled via specialized versions of the rules for handling $\mu f.\lambda x.e$ that ignore an unreferenced f.

[3] The `let`-polymorphism implied by a `let` construct can be simulated by intersection types.

$$
\begin{array}{rcl}
FV\mathrm{e} & : & \mathbf{LabExpr} \rightarrow \mathcal{P}(\mathbf{Var}) \\
FV_{\mathrm{e}}(ue^{l}) & = & FV_{\mathrm{ue}}(ue) \\
\\
FV\mathrm{ue} & : & \mathbf{UnLabExpr} \rightarrow \mathcal{P}(\mathbf{Var}) \\
FV_{\mathrm{ue}}(z) & = & \{z\} \\
FV_{\mathrm{ue}}(\mu f.\lambda x.e) & = & FV_{\mathrm{e}}(e) \setminus \{f, x\} \\
FV_{\mathrm{ue}}(e_1 \mathbin{@} e_2) & = & FV_{\mathrm{e}}(e_1) \cup FV_{\mathrm{e}}(e_2) \\
FV_{\mathrm{ue}}(c) & = & \emptyset \\
FV_{\mathrm{ue}}(\texttt{succ}\ e) & = & FV_{\mathrm{e}}(e) \\
FV_{\mathrm{ue}}(\texttt{if0}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3) & = & FV_{\mathrm{e}}(e_1) \cup FV_{\mathrm{e}}(e_2) \cup FV_{\mathrm{e}}(e_3) \\
FV_{\mathrm{ue}}(\texttt{bind}\ se\ \texttt{in}\ e) & = & FV_{\mathrm{se}}(se) \cup (FV_{\mathrm{e}}(e) \setminus dom(se)) \\
FV_{\mathrm{ue}}(\texttt{close}\ fn\ \texttt{in}\ se) & = & FV_{\mathrm{se}}(se) \cup (FV_{\mathrm{ue}}(fn) \setminus dom(se)) \\
\\
FV\mathrm{se} & : & \mathbf{SemEnv} \rightarrow \mathcal{P}(\mathbf{Var}) \\
FV_{\mathrm{se}}(se) & = & \bigcup_{v \in cod(se)} FV_{\mathrm{ue}}(v)
\end{array}
$$

Figure 2: Free variables in $\mathcal{L}$.

environment se as a partial function from **Var** to **Value**: if $z \in dom(se)$, then $se(z) = sv$, where $[z \mapsto sv]$ is the *rightmost* binding in sv; otherwise $se(z)$ is undefined. Note that $\mathcal{E}[z \mapsto sv](z')$ equals sv if $z = z'$ and equals $\mathcal{E}(z')$ otherwise. In later sections, we shall use the same environment notation presented here for environments mapping variables to entities other than values.

Expressions are annotated with labels that serve to identify them in the type and flow systems. There are two alternating classes of expressions: labeled expressions are unlabeled expressions annotated with labels, and unlabeled expressions have labeled expressions as immediate subexpressions[4]. We often write labels on constructors; e.g., we write $\lambda^l x.e$ for $(\lambda x.e)^l$ and $e_1 \mathbin{@}_l e_2$ for $(e_1 \mathbin{@} e_2)^l$. In examples, we use natural numbers for labels. The function lab : **UnLabExpr** $\rightarrow$ **Label** returns the label of a labeled expression; i.e., $lab(ue^l) = l$.

The free variables of labeled expressions ($FV_{\mathrm{e}}(e)$), unlabeled expressions ($FV_{\mathrm{ue}}(ue)$), and semantic environments ($FV_{\mathrm{se}}(se)$) are defined in Fig. 2. Note that the semantic environments in `bind` and `close` expressions act as binders for variables in the other components of these expressions.

An expression or environment containing free variables is said to be *open*; otherwise it is *closed*. An expression or environment containing `bind` or `close` expressions is said to be *impure*; otherwise it is *pure*. A pure expression e (resp. ue) is said to be *uniquely labeled* if each label occurs at most once in e (resp. ue). A pleasant consequence of this last property is that each subexpression of an expression e denotes a unique "position" within e. A program P is a pure, closed expression that is uniquely labeled.

Example 2.1. The program $\mathsf{P}_{2.1} = \left(\lambda^6 \mathtt{g}.\left(\left(\mathtt{g}^3 \mathbin{@}_2 \mathtt{g}^4\right) \mathbin{@}_1 0^5\right)\right) \mathbin{@}_0 \left(\lambda^8 \mathtt{x}.\mathtt{x}^7\right)$ shows the need for polyvariance: $\lambda^8 \mathtt{x}.\mathtt{x}^7$ is applied both to itself and to an integer. □

Example 2.2 (P&P, Sect.1.6). The following program $\mathsf{P}_{2.2}$ requires even more powerful polyvariance. (Assume that e_c is a closed expression denoting an integer.)

$$\mathsf{P}_{2.2} = \left(\lambda^6 \mathtt{g}.\mathtt{succ}^{13}\ \left(\left(\mathtt{g}^3 \mathbin{@}_2 \mathtt{g}^4\right) \mathbin{@}_1 0^5\right)\right) \mathbin{@}_0 \left(\mathtt{if0}^9\ e_c\ \mathtt{then}\ \left(\lambda^8 \mathtt{x}.\mathtt{x}^7\right)\ \mathtt{else}\ \left(\lambda^{12} \mathtt{y}.\lambda^{11} \mathtt{z}.\mathtt{z}^{10}\right)\right)$$

□

$\mathcal{L}$ is similar to the labeled languages studied by N&N and P&P. It differs from N&N's language only in the addition of `succ` and the omission of a `let` construct. It differs from

[4] Closures `close` fn `in` se are an exception to this rule; the immediate subexpression fn is *unlabelled*.

P&P's language only in its support for recursive functions and its use of `bind` and `close` as intermediate expressions used by the semantics.

In the formal development, it is necessary to refer to various syntactic entities occurring within a given expression or environment. Fig. 3 presents the definitions of the syntax manipulation functions we use later in the paper. The function call $\mathit{LabExps}_{\mathrm{e}}(e)$ returns a set of all labeled subexpressions occurring within the given expression e. Similarly, $\mathit{LabExps}_{\mathrm{ue}}(ue)$ returns the set of all labeled subexpressions occurring within the unlabeled expression ue, and $\mathit{LabExps}_{\mathrm{se}}(se)$ returns the set of all labeled subexpressions occurring within the semantic environment se. To reduce clutter in Fig. 3, we use the metavariable $\mathcal{X}$ to range over the symbols $\{e, ue, se\}$, and define $\mathbf{Domain}_e = \mathbf{LabExpr}$, $\mathbf{Domain}_{ue} = \mathbf{UnLabExpr}$, and $\mathbf{Domain}_{se} = \mathbf{SemEnv}$. Using $\mathcal{X}$, we can summarize the signatures of all three functions for labeled subexpressions via the pattern $\mathit{LabExps}_{\mathcal{X}} : \mathbf{Domain}_{\mathcal{X}} \to \mathcal{P}(\mathbf{LabExpr})$, where $\mathcal{P}(S)$ denotes the power set (set of subsets) of the set S. Similarly, there are three functions returning the *unlabeled* subexpressions of expressions and environments that are summarized by the signature pattern $\mathit{UnLabExps}_{\mathcal{X}} : \mathbf{Domain}_{\mathcal{X}} \to \mathcal{P}(\mathbf{UnLabExpr})$, and three functions returning the semantic environments occurring in expressions and environments that are summarized by the signature pattern $\mathit{Envs}_{\mathcal{X}} : \mathbf{Domain}_{\mathcal{X}} \to \mathcal{P}(\mathbf{SemEnv})$. $\mathit{LabExps}_{\mathcal{X}}$, $\mathit{UnLabExps}_{\mathcal{X}}$, and $\mathit{Envs}_{\mathcal{X}}$ are all defined in terms of auxiliary functions $\mathit{SF}_{\mathcal{X}}$ that return a triple ($\in$ **SubForms**) of all (1) labeled expressions (2) unlabeled expressions and (3) semantic environments encountered in a recursive descent of an expression or environment. The functions $\mathit{Labs}_{\mathcal{X}}$ return the labels occurring within a given syntactic entity. There are similar functions for returning the variables ($\mathit{Vars}_{\mathcal{X}}$), abstractions ($\mathit{Funs}_{\mathcal{X}}$), binders ($\mathit{Binds}_{\mathcal{X}}$), and closures ($\mathit{Clos}_{\mathcal{X}}$) within a given syntactic entity.

Definition 2.3 (Well-formedness of Expressions and Environments). A labeled expression e is *well-formed wrt. a program P*, written $\mathit{wf}^{P}_{\mathrm{e}}(e)$ iff all of the following conditions hold:

1. $\mathit{Vars}_{\mathrm{e}}(e) \subseteq \mathit{Vars}_{\mathrm{e}}(P)$
2. $\mathit{Labs}_{\mathrm{e}}(e) \subseteq \mathit{Labs}_{\mathrm{e}}(P)$
3. $\mathit{Funs}_{\mathrm{e}}(e) \subseteq \mathit{Funs}_{\mathrm{e}}(P)$
4. for all $bd \in \mathit{Binds}_{\mathrm{e}}(e)$, $FV_{\mathrm{ue}}(bd) = \emptyset$
5. for all $cl \in \mathit{Clos}_{\mathrm{e}}(e)$, $FV_{\mathrm{ue}}(cl) = \emptyset$
6. for all $se \in \mathit{Envs}_{\mathrm{e}}(e)$, $FV_{\mathrm{se}}(se) = \emptyset$

Well-formedness wrt. a program P for an unlabeled expression, written $\mathit{wf}^{P}_{\mathrm{ue}}(ue)$, is defined similarly to the above definition, except that e is replaced by ue and F_e is replaced by F_{ue} for F ranging over *Vars*, *Labs*, *Funs*, *Binds*, *Clos*, and *Envs*. A semantic environment se is well-formed wrt. a program P iff se is closed and $\mathit{wf}^{P}_{\mathrm{ue}}(v)$ for all $v \in \mathit{cod}(se)$. □

We say that an expression or environment is *well-formed* if there is some program P such that the expression or environment is well-formed wrt. a program P. Often, a particular program P will be implied from the context. We use E as a metavariable ranging over all closed, well-formed expressions.

There are several facts implied by the above definition that we often use later:

- For all programs P it holds that $\mathit{wf}^{P}_{\mathrm{e}}(P)$.
- Every subexpression or environment occurring within a well-formed expression or environment is itself necessarily well-formed.
- Because programs are necessarily pure, condition (3) implies that all abstractions occuring in a well-formed expression or environment have pure bodies.

$$
\begin{array}{rcl}
\mathbf{SubForms} & = & \mathcal{P}(\mathbf{LabExpr}) \times \mathcal{P}(\mathbf{UnLabExpr}) \times \mathcal{P}(\mathbf{SemEnv}) \\[1em]
\oplus & : & (\mathbf{SubForms} \times \mathbf{SubForms}) \rightarrow \mathbf{SubForms} \\
\langle e_1, ue_1, se_1 \rangle \oplus \langle e_2, ue_2, se_2 \rangle & = & \langle e_1 \ \cup \ e_2, ue_1 \ \cup \ ue_2, se_1 \ \cup \ se_2 \rangle \\[1em]
\mathit{SF}_{\mathrm{e}} & : & \mathbf{LabExpr} \rightarrow \mathbf{SubForms} \\
\mathit{SF}_{\mathrm{e}}(ue^l) & = & \left\langle \left\{ ue^l \right\}, \emptyset, \emptyset \right\rangle \oplus \mathit{SF}_{\mathrm{ue}}(ue) \\[1em]
\mathit{SF}_{\mathrm{ue}} & : & \mathbf{UnLabExpr} \rightarrow \mathbf{SubForms} \\
\mathit{SF}_{\mathrm{ue}}(ue) & = & \langle \emptyset, \{ue\}, \emptyset \rangle \oplus \mathit{SF}'_{\mathrm{ue}}(ue) \\[1em]
\mathit{SF}'_{\mathrm{ue}} & : & \mathbf{UnLabExpr} \rightarrow \mathbf{SubForms} \\
\mathit{SF}'_{\mathrm{ue}}(z) & = & \langle \emptyset, \emptyset, \emptyset \rangle \\
\mathit{SF}'_{\mathrm{ue}}(\mu f.\lambda x.e) & = & \mathit{SF}_{\mathrm{e}}(e) \\
\mathit{SF}'_{\mathrm{ue}}(e_1 \,@\, e_2) & = & \mathit{SF}_{\mathrm{e}}(e_1) \oplus \mathit{SF}_{\mathrm{e}}(e_2) \\
\mathit{SF}'_{\mathrm{ue}}(c) & = & \langle \emptyset, \emptyset, \emptyset \rangle \\
\mathit{SF}'_{\mathrm{ue}}(\mathtt{succ}\ e) & = & \mathit{SF}_{\mathrm{e}}(e) \\
\mathit{SF}'_{\mathrm{ue}}(\mathtt{if0}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3) & = & \mathit{SF}_{\mathrm{e}}(e_1) \oplus \mathit{SF}_{\mathrm{e}}(e_2) \oplus \mathit{SF}_{\mathrm{e}}(e_3) \\
\mathit{SF}'_{\mathrm{ue}}(\mathtt{bind}\ se\ \mathtt{in}\ e) & = & \mathit{SF}_{\mathrm{e}}(e) \oplus \mathit{SF}_{\mathrm{se}}(se) \\
\mathit{SF}'_{\mathrm{ue}}(\mathtt{close}\ \mathit{fn}\ \mathtt{in}\ se) & = & \mathit{SF}_{\mathrm{ue}}(\mathit{fn}) \oplus \mathit{SF}_{\mathrm{se}}(se) \\[1em]
\mathit{SF}_{\mathrm{se}} & : & \mathbf{SemEnv} \rightarrow \mathbf{SubForms} \\
\mathit{SF}_{\mathrm{se}}(se) & = & \langle \emptyset, \emptyset, se \rangle \oplus \left(\bigoplus_{v \in \mathit{cod}(se)} \mathit{SF}_{\mathrm{ue}}(\{v\}) \right)
\end{array}
$$

Let $\mathcal{X}$ range over the symbols $\{e, ue, se\}$ and define

$$
\begin{array}{rcl}
\mathbf{Domain}_e & = & \mathbf{LabExpr} \\
\mathbf{Domain}_{ue} & = & \mathbf{UnLabExpr} \\
\mathbf{Domain}_{se} & = & \mathbf{SemEnv} \\[1em]
\mathit{LabExps}_{\mathcal{X}} & : & \mathbf{Domain}_{\mathcal{X}} \rightarrow \mathcal{P}(\mathbf{LabExpr}) \\
\mathit{LabExps}_{\mathcal{X}}(\mathcal{X}) & = & a, \text{ where } \langle a, b, c \rangle = \mathit{SF}_{\mathcal{X}}(\mathcal{X}) \\[1em]
\mathit{UnLabExps}_{\mathcal{X}} & : & \mathbf{Domain}_{\mathcal{X}} \rightarrow \mathcal{P}(\mathbf{UnLabExpr}) \\
\mathit{UnLabExps}_{\mathcal{X}}(\mathcal{X}) & = & b, \text{ where } \langle a, b, c \rangle = \mathit{SF}_{\mathcal{X}}(\mathcal{X}) \\[1em]
\mathit{Envs}_{\mathcal{X}} & : & \mathbf{Domain}_{\mathcal{X}} \rightarrow \mathcal{P}(\mathbf{SemEnv}) \\
\mathit{Envs}_{\mathcal{X}}(\mathcal{X}) & = & c, \text{ where } \langle a, b, c \rangle = \mathit{SF}_{\mathcal{X}}(\mathcal{X}) \\[1em]
\mathit{Labs}_{\mathcal{X}} & : & \mathbf{Domain}_{\mathcal{X}} \rightarrow \mathcal{P}(\mathbf{Label}) \\
\mathit{Labs}_{\mathcal{X}}(\mathcal{X}) & = & \left\{ l \mid \exists ue.\ ue^l \in \mathit{LabExps}_{\mathcal{X}}(\mathcal{X}) \right\} \\[1em]
\mathit{Vars}_{\mathcal{X}} & : & \mathbf{Domain}_{\mathcal{X}} \rightarrow \mathcal{P}(\mathbf{Var}) \\
\mathit{Vars}_{\mathcal{X}}(\mathcal{X}) & = & \{ z \mid z \in \mathit{UnLabExps}_{\mathcal{X}}(\mathcal{X}) \} \\[1em]
\mathit{Funs}_{\mathcal{X}} & : & \mathbf{Domain}_{\mathcal{X}} \rightarrow \mathcal{P}(\mathbf{Abstr}) \\
\mathit{Funs}_{\mathcal{X}}(\mathcal{X}) & = & \{ \mathit{fn} \mid \mathit{fn} \in \mathit{UnLabExps}_{\mathcal{X}}(\mathcal{X}) \} \\[1em]
\mathit{Binds}_{\mathcal{X}} & : & \mathbf{Domain}_{\mathcal{X}} \rightarrow \mathcal{P}(\mathbf{Binder}) \\
\mathit{Binds}_{\mathcal{X}}(\mathcal{X}) & = & \{ bd \mid bd \in \mathit{UnLabExps}_{\mathcal{X}}(\mathcal{X}) \} \\[1em]
\mathit{Clos}_{\mathcal{X}} & : & \mathbf{Domain}_{\mathcal{X}} \rightarrow \mathcal{P}(\mathbf{Closure}) \\
\mathit{Clos}_{\mathcal{X}}(\mathcal{X}) & = & \{ cl \mid cl \in \mathit{UnLabExps}_{\mathcal{X}}(\mathcal{X}) \}
\end{array}
$$

Figure 3: Definitions of subexpression functions for $\mathcal{L}$.

- Condition (5) implies that all values in a well-formed expression or environment are necessarily closed.

The following lemma is handy for showing the well-formedness of an expression in terms of the well-formedness of its immediate components.

Lemma 2.4 (Necessary and Sufficient Conditions for Well-formedness). *If $e = ue^l$, then $wf^P_{\mathrm{e}}(e)$ holds if and only if (1) $l \in Labs_{\mathrm{e}}(P)$ and (2) ue satisfies the following requirements based on the form of ue:*

- $\underline{ue = z}$ *implies* $z \in Vars_{\mathrm{e}}(P)$;
- $\underline{ue = fn}$ *implies* $fn \in Funs_{\mathrm{e}}(P)$;
- $\underline{ue = e_1 \mathbin{@} e_2}$ *implies* $wf^P_{\mathrm{e}}(e_1)$ *and* $wf^P_{\mathrm{e}}(e_2)$;
- $\underline{ue = c}$ *implies no requirements;*
- $\underline{ue = \mathtt{succ}\ e}$ *implies* $wf^P_{\mathrm{e}}(e)$;
- $\underline{ue = \mathtt{if0}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e}$ *implies* $wf^P_{\mathrm{e}}(e_1)$, $wf^P_{\mathrm{e}}(e_2)$, *and* $wf^P_{\mathrm{e}}(e_3)$;
- $\underline{ue = \mathtt{bind}\ se\ \mathtt{in}\ e}$ *implies* $wf^P_{\mathrm{e}}(e)$, $wf^P_{\mathrm{se}}(se)$, *and* $FV_{\mathrm{ue}}(ue) = \emptyset$;
- $\underline{ue = \mathtt{close}\ fn\ \mathtt{in}\ se}$ *implies* $fn \in Funs_{\mathrm{e}}(P)$, $wf^P_{\mathrm{se}}(se)$, *and* $FV_{\mathrm{ue}}(ue) = \emptyset$.

□

Proof. Follows easily from Def. 2.3. □

2.2 Semantics

Like N&N, but unlike P&P, we specify the meaning of programs using an environment-based small-step semantics (Fig. 4). Each step of the semantics is expressed as a judgement of the form $se \vdash e \Rightarrow e'$, which says that e rewrites to e' in one step relative to a semantic environment se. Note that this rewriting is deterministic. The binary relation $se \vdash \_ \Rightarrow^* \_$ is the reflexive, transitive closure of $se \vdash \_ \Rightarrow \_$. We abbreviate $\epsilon \vdash e \Rightarrow e'$ as $e \Rightarrow e'$.

Our rules are adapted from those in N&N; we deviate from theirs in that we do not truncate the semantic environment se in the rule (*fun*), cf. the discussion in Sec. 4 of our rule [*fun*] in Fig. 10. There are a few aspects of the rules that are worth highlighting. The (*fun*) rule makes a closure from a labeled abstraction by pairing the *unlabeled* abstraction with the current semantic environment. The (app_v) rule evaluates the body of the applied closure relative to its environment, extended with bindings for the μ-bound variable f and the λ-bound variable x. The (*bind*) and ($bind_v$) rules say that in $se \vdash \mathtt{bind}^l\ se_1\ \mathtt{in}\ e \Rightarrow e'$, the body e of the $\mathtt{bind}$ is evaluated relative to the semantic environment se_1, and not to se, which is ignored.

Example 2.5. Consider the evaluation of program $\mathsf{P}_{2.2}$ from Ex. 2.2. We introduce the following abbreviations:

$$\begin{array}{rcl}
ue_{\lambda \mathtt{g}} & = & \lambda \mathtt{g}.\mathtt{succ}^{13}\left(\left(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4\right) \mathbin{@_1} 0^5\right) \\
ue_{\lambda \mathtt{z}} & = & \lambda \mathtt{z}.\mathtt{z}^{10} \\
ue_{\lambda \mathtt{y}} & = & \lambda \mathtt{y}.ue^{11}_{\lambda \mathtt{z}} \\
cl_{\lambda \mathtt{y}} & = & \mathtt{close}\ ue_{\lambda \mathtt{y}}\ \mathtt{in}\ \epsilon \\
cl_{\lambda \mathtt{z}} & = & \mathtt{close}\ ue_{\lambda \mathtt{z}}\ \mathtt{in}\ se_{\mathtt{y}} \\
se_{\mathtt{g}} & = & \epsilon[\mathtt{g} \mapsto cl_{\lambda \mathtt{y}}] \\
se_{\mathtt{y}} & = & \epsilon[\mathtt{y} \mapsto cl_{\lambda \mathtt{y}}] \\
se_{\mathtt{z}} & = & se_{\mathtt{y}}[\mathtt{z} \mapsto 0]
\end{array}$$

$(var)\quad se \vdash z^l \Rightarrow sv^l$, if $sv = se(z)$

$(fun)\quad se \vdash \mu f.\lambda^l x.e \Rightarrow \texttt{close}^l\ \mu f.\lambda x.e\ \texttt{in}\ se$

$$(app_l)\quad \frac{se \vdash e_1 \Rightarrow e_1'}{se \vdash e_1 \mathbin{@_l} e_2 \Rightarrow e_1' \mathbin{@_l} e_2}$$

$$(app_r)\quad \frac{se \vdash e_2 \Rightarrow e_2'}{se \vdash cl_1^{l_1} \mathbin{@_l} e_2 \Rightarrow cl_1^{l_1} \mathbin{@_l} e_2'}$$

$$(app_v)\quad \begin{array}{l} se \vdash \left(\texttt{close}^{l_1}\ fn_1\ \texttt{in}\ se_1\right) \mathbin{@_l} sv_2^{l_2} \\ \quad \Rightarrow \texttt{bind}^l\ se_1[f \mapsto (\texttt{close}\ fn_1\ \texttt{in}\ se_1)][x \mapsto sv_2]\ \texttt{in}\ e_1, \\ \qquad \text{where } fn_1 = \mu f.\lambda x.e_1 \end{array}$$

$$(succ)\quad \frac{se \vdash e_1 \Rightarrow e_1'}{se \vdash \texttt{succ}^l\ e_1 \Rightarrow \texttt{succ}^l\ e_1'}$$

$(succ_v)\quad se \vdash \texttt{succ}^l\ c^{l_0} \Rightarrow c_1^l$, where $c_1 = c + 1$

$$(if)\quad \frac{se \vdash e_0 \Rightarrow e_0'}{se \vdash \texttt{if0}^l\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Rightarrow \texttt{if0}^l\ e_0'\ \texttt{then}\ e_1\ \texttt{else}\ e_2}$$

$(if_0)\quad se \vdash \texttt{if0}^l\ 0^{l_0}\ \texttt{then}\ ue_1^{l_1}\ \texttt{else}\ ue_2^{l_2} \Rightarrow ue_1^l$

$(if_>)\quad se \vdash \texttt{if0}^l\ c^{l_0}\ \texttt{then}\ ue_1^{l_1}\ \texttt{else}\ ue_2^{l_2} \Rightarrow ue_2^l$, if $c \neq 0$

$$(bind)\quad \frac{se_1 \vdash e_1 \Rightarrow e_1'}{se \vdash \texttt{bind}^l\ se_1\ \texttt{in}\ e_1 \Rightarrow \texttt{bind}^l\ se_1\ \texttt{in}\ e_1'}$$

$(bind_v)\quad se \vdash \texttt{bind}^l\ se_1\ \texttt{in}\ sv^{l_1} \Rightarrow sv^l$

Figure 4: Semantics of $\mathfrak{L}$.

Suppose that $e_c = ue_c^{14}$ and $e_c \Rightarrow^* 17$. Then we have the following evaluation sequence starting at $\mathsf{P}_{2.2}$:

$$\begin{array}{l}
\mathsf{P}_{2.2} = ue_{\lambda\mathtt{g}}^6 \mathbin{@_0} \left(\texttt{if0}^9\ e_c\ \texttt{then}\ \left(\lambda^8\mathtt{x.x}^7\right)\ \texttt{else}\ ue_{\lambda\mathtt{y}}^{12}\right) \\
\Rightarrow \left(\texttt{close}^6\ ue_{\lambda\mathtt{g}}\ \texttt{in}\ \epsilon\right) \mathbin{@_0} \left(\texttt{if0}^9\ e_c\ \texttt{then}\ \left(\lambda^8\mathtt{x.x}^7\right)\ \texttt{else}\ ue_{\lambda\mathtt{y}}^{12}\right) \\
\Rightarrow^* \left(\texttt{close}^6\ ue_{\lambda\mathtt{g}}\ \texttt{in}\ \epsilon\right) \mathbin{@_0} \left(\texttt{if0}^9\ 17^{14}\ \texttt{then}\ \left(\lambda^8\mathtt{x.x}^7\right)\ \texttt{else}\ ue_{\lambda\mathtt{y}}^{12}\right) \\
\Rightarrow \left(\texttt{close}^6\ ue_{\lambda\mathtt{g}}\ \texttt{in}\ \epsilon\right) \mathbin{@_0} ue_{\lambda\mathtt{y}}^9 \\
\Rightarrow \left(\texttt{close}^6\ ue_{\lambda\mathtt{g}}\ \texttt{in}\ \epsilon\right) \mathbin{@_0} cl_{\lambda\mathtt{y}}^9 \\
\Rightarrow \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ \texttt{succ}^{13}\ \left(\left(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4\right) \mathbin{@_1} 0^5\right) \\
\Rightarrow^* \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ \texttt{succ}^{13}\ \left(\left(cl_{\lambda\mathtt{y}}^3 \mathbin{@_2} cl_{\lambda\mathtt{y}}^4\right) \mathbin{@_1} 0^5\right) \\
\Rightarrow \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ \texttt{succ}^{13}\ \left(\left(\texttt{bind}^2\ se_{\mathtt{y}}\ \texttt{in}\ ue_{\lambda\mathtt{z}}^{11}\right) \mathbin{@_1} 0^5\right) \\
\Rightarrow \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ \texttt{succ}^{13}\ \left(\left(\texttt{bind}^2\ se_{\mathtt{y}}\ \texttt{in}\ cl_{\lambda\mathtt{z}}^{11}\right) \mathbin{@_1} 0^5\right) \\
\Rightarrow \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ \texttt{succ}^{13}\ \left(cl_{\lambda\mathtt{z}}^2 \mathbin{@_1} 0^5\right) \\
\Rightarrow \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ \texttt{succ}^{13}\ \left(\texttt{bind}^1\ se_{\mathtt{z}}\ \texttt{in}\ \mathtt{z}^{10}\right) \\
\Rightarrow \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ \texttt{succ}^{13}\ \left(\texttt{bind}^1\ se_{\mathtt{z}}\ \texttt{in}\ 0^{10}\right) \\
\Rightarrow \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ \texttt{succ}^{13}\ 0^1 \\
\Rightarrow \texttt{bind}^0\ se_{\mathtt{g}}\ \texttt{in}\ 1^{13} \\
\Rightarrow 1^0
\end{array}$$

□

Theorem 2.6 (Properties of Evaluation). *Given program P, labeled expression e, and semantic environment se, suppose that $wf_{\mathrm{se}}^P(se)$, $wf_{\mathrm{e}}^P(e)$, and $FV_{\mathrm{e}}(e) \subseteq dom(se)$. Then*

$se \vdash e \Rightarrow^* e'$ implies the following:

1. $lab(e') = lab(e)$
2. $FV_e(e') \subseteq FV_e(e)$
3. $wf_e^P(e')$

$\square$

Proof. It is sufficient to show that each of the properties holds for one evaluation step; the multi-step properties then follow via an easy induction. Property (1) follows from the fact that each rule in Fig. 4 preserves the top-level label of an expression. Properties (2) and (3) follow together from an induction in the derivation of $se \vdash e \Rightarrow e'$. The important cases are shown below, where we implicitly rely on Lem. 2.4 for showing the well-formednes of e'. Note that $wf_e^P(e)$ implies in all cases that the top-level label $l \in Labs_e(P)$.

<u>*(var)*.</u> In this case, $e = z^l$ and $e' = sv^l$ where $sv = se(z)$. From $wf_{se}^P(se)$ we see that se and therefore also $se(z)$ is closed, establishing (2), and that $wf_{ue}^P(sv)$, establishing (3).

<u>*(fun)*.</u> In this case, $e = \mu f.\lambda^l x.e_1$ and $e' = \texttt{close}^l\ \mu f.\lambda x.e_1\ \texttt{in}\ se$. By assumption, se is well-formed (and therefore closed) and $FV_e(e) \subseteq dom(se)$. Thus e' is closed, establishing not only (2) but (since $wf_e^P(e)$ implies $\mu f.\lambda x.e_1 \in Funs_e(P)$) also (3).

<u>*(app$_v$)*.</u> In this case, $e = cl^{l_1} @_l sv_2^{l_2}$ where $cl = \texttt{close}\ fn_1\ \texttt{in}\ se_1$ and $fn_1 = \mu f.\lambda x.e_1$, and $e' = \texttt{bind}^l\ se'\ \texttt{in}\ e_1$ where $se' = se_1[f \mapsto cl][x \mapsto sv_2]$. Since e is well-formed, so are the subexpressions cl, sv_2, and e_1, and the environment se_1, in particular it holds that cl and sv_2 are closed. This enables us to deduce that se' is well-formed, and that $FV_e(e_1) \subseteq (dom(se_1) \cup \{f, x\}) = dom(se')$. Thus the resulting bind expression is closed, establishing not only (2) but also (3).

<u>*(bind)*.</u> In this case, $e = \texttt{bind}^l\ se_1\ \texttt{in}\ e_1$ and $e' = \texttt{bind}^l\ se_1\ \texttt{in}\ e_1'$ where $se \vdash e \Rightarrow e'$ because $se_1 \vdash e_1 \Rightarrow e_1'$. From $wf_e^P(e)$ we infer that $wf_e^P(e_1)$, $wf_{se}^P(se_1)$, and $FV_e(e_1) \subseteq dom(se_1)$. We can thus apply the induction hypothesis to infer that $FV_e(e_1') \subseteq FV_e(e_1)$ and that $wf_e^P(e_1')$. But this clearly enables us to establish (2) and (3).

<u>*(bind$_v$)*.</u> In this case, $e = \texttt{bind}^l\ se_1\ \texttt{in}\ sv^{l_1}$ and $e' = sv^l$. Since e is well-formed, the subexpression sv is well-formed, and so is sv^l, establishing (3). Since the value sv is necessarily closed, (2) is established.

$\square$

Corollary 2.7 (Program Evaluation). *If P is a program and $P \Rightarrow^* E$, then E is closed, $lab(E) = lab(P)$, and $wf_e^P(E)$. The fact that E is closed and well-formed justifies the use of the metavariable E in this context.* $\square$

REMARK 2.8. The fact that $\Rightarrow$ preserves the top-level labels of expressions, unlike what is the case for P&P's evaluation rules, seems to be essential (cf. Sec. 4.2) for establishing that the flow analysis in Sec. 4 is indeed a "closure analysis". $\square$

REMARK 2.9. [NN98] have argued that an environment-based semantics is more suitable for proving the semantic soundness of a flow analysis than a semantics based on explicit substitutions. This methodological claim is confirmed by the fact that we are able to show a "subject evaluation" result for our flow logic that is inherited from N&N's approach. P&P choose, however, a substitution semantics, so as to avoid technical difficulties involved in proving subject evaluation for their type system. See Sec. 3 for how we address this issue.

$\square$

REMARK 2.10. A consequence of the label preservation property of evaluation is that the label of abstractions and closures are *not* preserved by evaluation. For example, in the

$t \in \textbf{ElementaryType} ::= \texttt{int} \mid \bigwedge_{i\in I}\{K_i : u_i \to u'_i\}$
$u \in \textbf{UnionType} ::= \bigvee_{i\in I}\{q_i : t_i\}$
$K \in \mathcal{P}(\textbf{ITag}) \setminus \{\emptyset\}$
$k \in \textbf{ITag}$
$q \in \textbf{UTag}$

Figure 5: Syntax of types in the type system $\mathcal{TS}$ for language $\mathcal{L}$. **ITag** and **UTag** are unspecified finite sets of tags for intersection and union types, respectively.

evaluation sequence shown in Ex. 2.5, the abstraction $ue_{\lambda\texttt{y}}$ has the label 12 in the original program $\mathsf{P}_{2.2}$, but this label is stripped when the abstraction is paired with the empty environment to form the closure $cl_{\lambda\texttt{y}}$. This closure is initially given label 9 (the label of the `if0` in $\mathsf{P}_{2.2}$), but this label is stripped when $cl_{\lambda\texttt{y}}$ is bound to `g` in $se_{\texttt{g}}$, and substituting for `g` in the body of $ue_{\lambda\texttt{g}}$ leads to copies of $cl_{\lambda\texttt{y}}$ labeled 3 and 4.

However, the label of the *body* of an abstraction *is* preserved via evaluation, even when the abstraction is copied. For instance, all copies of $ue_{\lambda\texttt{y}}$ that appear in the evaluation sequence of Ex. 2.5 have a body labeled 11. This property is a consequence of requiring that every (unlabeled) abstraction in a program well-formed with respect to P must occur in P itself (condition (3) of Def. 2.3).

□

3 The Type System

3.1 Type Syntax

In this section we present the type system $\mathcal{TS}$ for our language $\mathcal{L}$. The types of $\mathcal{TS}$ are built from base types, function types, intersection types, and union types as shown in Fig. 5.

An *elementary type* t is either the integer type `int` or an intersection type of the form $\bigwedge_{i\in I}\{K_i : u_i \to u'_i\}$, where:

- I is a (possibly empty) finite index set,
- each u_i and u'_i is a union type,
- each K_i, known as an *I-tagset*, is a non-empty finite set of *I-tags* from **ITag**, and
- the *I-tagsets* for t are pairwise disjoint: i.e., $K_i \cap K_j \neq \emptyset$ implies $i = j$.

We write $dom(t)$ for $\cup_{i\in I} K_i$, which is undefined if $t = \texttt{int}$. Intuitively, an entity that has the type $\bigwedge_{i\in I}\{K_i : u_i \to u'_i\}$ denotes a function that, *for all* $i \in I$, maps values of type u_i into values of type u'_i. For this reason, intersection types can be viewed as finite analogs of universal types.

A union type u has the form $\bigvee_{i\in I}\{q_i : t_i\}$, where:

- I is a (possibly empty) finite index set,
- each t_i is an elementary type, and
- the q_i are distinct *U-tags* in **UTag**.

We write $dom(u)$ for $\cup_{i\in I}\{q_i\}$, and $u.q = t$ if there exists $i \in I$ such that $q = q_i$ and $t = t_i$. We assume that for all $i \in I$ it holds that $t_i = \texttt{int}$ iff $q_i = q_{\text{int}}$ where q_{int} is a distinguished U-tag; this reflects that the U-tags are of interest only for function types, not for base types. Intuitively, if an expression e has the union type $\bigvee_{i\in I}\{q_i : t_i\}$, then *there exists* an $i \in I$ such that e has the elementary type t_i. For this reason, union types can be viewed as finite analogs of existential types. The analogy between intersection/union types and universal/existential types is explored in [WDMT0X].

If $I = \{1, \dots, n\}$, where $n \geq 0$, we write $\bigvee(q_1 : t_1, \dots, q_n : t_n)$ for $\bigvee_{i \in I}\{q_i : t_i\}$ and write $\bigwedge(K_1 : u_1 \to u'_1, \dots, K_n : u_n \to u'_n)$ for $\bigwedge_{i \in I}\{K_i : u_i \to u'_i\}$. We write u_{int} for $\bigvee(q_{\text{int}} : \texttt{int})$.

Example 3.1. The following are types that will be used later in the typing of $\mathsf{P}_{2.1}$ from Ex. 2.1:

$$\begin{array}{rcl} u'_{\mathtt{x}} & = & \bigvee(q_{\mathtt{x}} : \bigwedge(\{1\} : u_{\text{int}} \to u_{\text{int}})) \\ u_{\mathtt{x}} & = & \bigvee(q_{\mathtt{x}} : \bigwedge(\{1\} : u_{\text{int}} \to u_{\text{int}}, \{2\} : u'_{\mathtt{x}} \to u'_{\mathtt{x}})) \\ u_{\mathtt{g}} & = & \bigvee(q_{\mathtt{g}} : \bigwedge(\{0\} : u_{\mathtt{x}} \to u_{\text{int}})) \end{array}$$

We shall see that manifest abstractions are always given a union type with a single intersection type as a component, like those above. The type $u'_{\mathtt{x}}$ is the type of a function created at source site $q_{\mathtt{x}}$ that flows to sink site 1 and maps integers to integers; we will call such a function an *i2i* function. The type $u_{\mathtt{x}}$ denotes a function created at source site $q_{\mathtt{x}}$ that maps integers to integers at sink site 1 and maps *i2i* functions to *i2i* functions at sink site 2. The types $u'_{\mathtt{x}}$ and $u_{\mathtt{x}}$ are two of the many types that can be given to the identity function $e_{\lambda\mathtt{x}} = \lambda^8\mathtt{x}.\mathtt{x}^7$. When supplied with a function $\mathtt{g}$ of type $u_{\mathtt{x}}$, the function $\lambda^6\mathtt{g}.\left(\left(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4\right) \mathbin{@_1} 0^5\right)$ returns an integer, so it can be given the type $u_{\mathtt{g}}$. In this case, the I-tags 1 and 2 in $u'_{\mathtt{x}}$ and $u_{\mathtt{x}}$ correspond directly to the labels on the application sites in $\lambda^6\mathtt{g}.\left(\left(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4\right) \mathbin{@_1} 0^5\right)$, but we shall see that in general the I-tags can be taken from an arbitrary finite set **ITag** that does not have a one-to-one correspondence with the application site labels. □

Example 3.2. The following are types that, in conjunction with those from the previous example, will be used later in the typing of $\mathsf{P}_{2.2}$ from Ex. 2.2:

$$\begin{array}{rcl} u_{\mathtt{z}} & = & \bigvee(q_{\mathtt{z}} : \bigwedge(\{1\} : u_{\text{int}} \to u_{\text{int}})) \\ u'_{\mathtt{y}} & = & \bigvee(q_{\mathtt{y}} : \bigwedge()) \\ u_{\mathtt{y}} & = & \bigvee(q_{\mathtt{y}} : \bigwedge(\{3\} : u'_{\mathtt{y}} \to u_{\mathtt{z}})) \\ u_{\mathtt{xy}} & = & \bigvee(q_{\mathtt{x}} : \bigwedge(\{1\} : u_{\text{int}} \to u_{\text{int}}, \{2\} : u'_{\mathtt{x}} \to u'_{\mathtt{x}}), q_{\mathtt{y}} : \bigwedge(\{3\} : u'_{\mathtt{y}} \to u_{\mathtt{z}})) \end{array}$$

Suppose, as in Ex. 2.2, that the function $e_{\lambda\mathtt{y}} = \lambda^{12}\mathtt{y}.\lambda^{11}\mathtt{z}.\mathtt{z}^{10}$ can be applied to itself, and the result of this application can be applied to an integer. Then $\lambda^{11}\mathtt{z}.\mathtt{z}^{10}$ can be given the *i2i* type $u_{\mathtt{z}}$. Loosely speaking, in an application $e_{\lambda\mathtt{y}} \mathbin{@} e_{\lambda\mathtt{y}}$, the right occurrence of $e_{\lambda\mathtt{y}}$ can be given the type $u'_{\mathtt{y}}$; the empty intersection type indicates that this function is never applied. The left occurrence of $e_{\lambda\mathtt{y}}$ can be given the type $u_{\mathtt{y}}$, because it maps the right occurence to an *i2i* function. The expression $\mathtt{if0}^9\ e_c\ \mathtt{then}\ \left(\lambda^8\mathtt{x}.\mathtt{x}^7\right)\ \mathtt{else}\ \left(\lambda^{12}\mathtt{y}.\lambda^{11}\mathtt{z}.\mathtt{z}^{10}\right)$ can have the type of either $e_{\lambda\mathtt{x}}$ or $e_{\lambda\mathtt{y}}$, so it can be given the union type $u_{\mathtt{xy}}$. □

Grammars are usually interpreted inductively, but we intend that the one in Fig. 5 should be viewed co-inductively. That is, types are regular (possibly infinite) trees formed according to the given specification. Two types are considered equal if their infinite unwindings are equal (modulo renaming of the index sets I).

Example 3.3. Suppose that $q_1 = q'_4$ and $q_2 = q'_3$ and t_1 equals t'_4 and t_2 equals t'_3. Then $\bigvee_{i \in \{1,2\}}\{q_i : t_i\}$ equals $\bigvee_{i \in \{3,4\}}\{q'_i : t'_i\}$. □

Our types are similar to those in P&P except for the presence of tags. As we shall see, such tags serve as witnesses for existential quantifiers in the subtyping relation and play crucial roles in the faithfulness of our flow/type correspondence. U-tags track the "source" of each intersection type (a function in the 0-CFA case, but more generally an abstract closure) and, like Heintze's abstraction labels, help to avoid the precision-losing merging seen in P&P's type-to-flow translation (cf. Sec. 1). I-tagsets track the "sinks" of each arrow type (an application site in 1-CFA, but more generally an abstract application context) and help to avoid unnecessary recursive types in the flow-to-type translation.

Note that each I-tag (typically) designates one sink to which an intersection type component (i.e., a function) can flow. In general, a single function can flow to many sinks, and many functions can flow to the same sink. This is why each intersection type component is tagged with a *set* of I-tags. In contrast, there is no need for sets of U-tags, because each U-tag (typically) designates one source, and sources are combined via union types. The disjointness of I-tagsets is a consequence of the fact that two functions that flow to the same sink and have the same type can be merged into a single a intersection component.

Note that our intersection and union types, unlike those of P&P, are not associative, commutative, or idempotent (ACI) due to the presence of U-tags and I-tagsets. For example, while $\bigvee(q_1 : t_1, q_2 : t_2)$ equals $\bigvee(q_2 : t_2, q_1 : t_1)$ it does not equal $\bigvee(q_1 : t_2, q_2 : t_1)$.

3.2 Subtyping

We define an ordering $\leq_u$ on union types and an ordering $\leq_t$ on elementary types, where $u \leq_u u'$ means that u' is less precise than u and similarly for $\leq_t$. To capture the intuition that something of type t_1 has one of the types t_1 or t_2, $\leq_u$ should satisfy $\bigvee(q_1 : t_1) \leq_u \bigvee(q_1 : t_1, q_2 : t_2)$. For $\leq_t$, we want to capture the following intuition: a function that can be assigned both types $u_1 \to u'_1$ and $u_2 \to u'_2$ also

- can be assigned one of them, i.e., for $i \in \{1, 2\}$, $\bigwedge(K_1 : u_1 \to u'_1, K_2 : u_2 \to u'_2) \leq_t \bigwedge(K_i : u_i \to u'_i)$;
- can be assigned a function type that "covers" both, i.e., $\bigwedge(K_1 : u_1 \to u'_1, K_2 : u_2 \to u'_2) \leq_t \bigwedge(K_1 \cup K_2 : u_{12} \to u'_{12})$ where any value having type u_{12} also has one of the types u_1 or u_2, and where any value having one of the types u'_1 or u'_2 also has type u'_{12}. For then a function that for all $i \in \{1, 2\}$ maps values of type u_i into values of type u'_i surely also will map a value of type u_{12} into a value of type u'_{12}.

The following mutually recursive specification of $\leq_u$ and $\leq_t$ formalizes the above considerations:

$\bigvee_{i\in I}\{q_i : t_i\} \leq_u \bigvee_{j\in J}\{q'_j : t'_j\}$
iff for all $i \in I$ there exists $j \in J$ such that $q_i = q'_j$ and $t_i \leq_t t'_j$

$\texttt{int} \leq_t \texttt{int}$

$\bigwedge_{i\in I}\{K_i : u_i \to u'_i\} \leq_t \bigwedge_{j\in J}\{K'_j : u''_j \to u'''_j\}$
iff for all $j \in J$ there exists $I_0 \subseteq I$ such that
$K'_j = \cup_{i\in I_0} K_i$ and $\forall i \in I_0.\ u'_i \leq_u u'''_j$ and
$\forall q \in \mathit{dom}(u''_j).\ \exists i \in I_0.\ q \in \mathit{dom}(u_i)$ and $u''_j.q \leq_t u_i.q$.

The above specification is not yet a *definition* of $\leq_u$ and $\leq_t$, since types may be infinite. However, it gives rise to a monotone functional $\mathcal{H}$ on a complete lattice[5]. We then define $\leq_u$ and $\leq_t$ as the (components of the) *greatest*[6] fixed point of this functional.

A proof by coinduction (given in Appendix A) yields:

Lemma 3.4. *The relations $\leq_u$ and $\leq_t$ are reflexive and transitive.* □

Observe that if $t \leq_t t'$, then $\mathit{dom}(t') \subseteq \mathit{dom}(t)$, and that if $t = \bigwedge_{i\in I}\{K_i : u_i \to u'_i\}$ and $t' = \bigwedge_{i\in I'}\{K_i : u_i \to u'_i\}$ with $I' \subseteq I$, then $t \leq_t t'$.

Our subtyping relation differs from P&P's in several ways. The U-tags and I-tags serve as "witnesses" for the existential quantifiers present in the specification, reducing the need

[5] The elements of which are (Q_u, Q_t), with Q_u a relation on union types and Q_t a relation on elementary types, and the ordering of which is pointwise subset inclusion.

[6] To motivate this choice, first note that the least fixed point is not even reflexive on infinite types. Second, even if we restricted our attention to reflexive and transitive relations, the least fixed point would not allow us to deduce $u_2 \leq_u u_1$ where the regular union types u_1 and u_2 are given by $u_1 = \bigvee(0 : \bigwedge(\{1\} : u_{\mathrm{int}} \to u_1))$ and $u_2 = \bigvee(0 : \bigwedge(\{1\} : u_{\mathrm{int}} \to u_2, \{2\} : u_{\mathrm{int}} \to u_{\mathrm{int}}))$.

$$
\begin{array}{lll}
[\mathsf{var}] & A \vdash z^l : u & \text{if } A(z) \leq_u u \\[6pt]
[\mathsf{fun}]^{(q:t)} & \dfrac{\forall k \in K : A[f \mapsto u''_k][x \mapsto u_k] \vdash e : u'_k}{A \vdash \mu f.\lambda^l x.e : u} & \begin{array}{l}\text{if } t = \bigwedge_{k \in K}\{\{k\} : u_k \rightarrow u'_k\} \\ \text{and } \bigvee(q:t) \leq_u u \\ \text{and } \forall k \in K.\ \bigvee(q:t) \leq_u u''_k\end{array} \\[6pt]
[\mathsf{app}]^{w^@} & \dfrac{A \vdash e_1 : u_1 \quad A \vdash e_2 : u_2}{A \vdash e_1 @_l e_2 : u} & \text{if } \forall q \in \mathit{dom}(u_1).\, u_1.q \leq_t \bigwedge(w^@(q) : u_2 \rightarrow u) \\[6pt]
[\mathsf{con}] & A \vdash c^l : u & \text{if } u_{\mathrm{int}} \leq_u u \\[6pt]
[\mathsf{suc}] & \dfrac{A \vdash e_1 : u_1}{A \vdash \mathtt{succ}^l\, e_1 : u} & \text{if } u_1 \leq_u u_{\mathrm{int}} \leq_u u \\[6pt]
[\mathsf{if}] & \dfrac{A \vdash e_0 : u_0 \quad A \vdash e_1 : u_1 \quad A \vdash e_2 : u_2}{A \vdash \mathtt{if0}^l\, e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 : u} & \text{if } u_0 \leq_u u_{\mathrm{int}} \text{ and } u_1 \leq_u u \text{ and } u_2 \leq_u u \\[6pt]
[\mathsf{bind}] & \dfrac{se \bowtie A' \quad A' \vdash e : u'}{A \vdash \mathtt{bind}^l\, se\ \mathtt{in}\ e : u} & \text{if } u' \leq_u u \\[6pt]
[\mathsf{clos}] & \dfrac{se \bowtie A' \quad A' \vdash \mathit{fn} : u}{A \vdash \mathtt{close}^l\, \mathit{fn}\ \mathtt{in}\ se : u} &
\end{array}
$$

Figure 6: The typing rules

for search during type checking. Moreover, our ordering seems more natural that the P&P's $\leq_1$, which has the rather odd property that if $\vee(T_1, T_2) \leq_1 \vee(T_3, T_4)$ (with the T_i's all distinct), then either $\vee(T_1, T_2) \leq_1 T_3$ or $\vee(T_1, T_2) \leq_1 T_4$, and which is in fact not a congruence: to see this, take some incomparable σ_1 and σ_2 and note that

$$\wedge(\sigma_1 \rightarrow \sigma_1, \sigma_2 \rightarrow \sigma_1) \leq_1 \sigma_1 \rightarrow \sigma_1 \text{ and } \wedge(\sigma_1 \rightarrow \sigma_2, \sigma_2 \rightarrow \sigma_2) \leq_1 \sigma_2 \rightarrow \sigma_2$$

but the union of the left hand sides is not $\leq_1$ the union of the right hand sides.

3.3 Typing Rules

A *typing* T for a program P is a tuple $\langle P, IT_T, UT_T, D_T \rangle$, where IT_T is a finite non-empty set of I-tags, UT_T is a finite set of U-tags, and D_T is a derivation of $\epsilon \vdash P : u$ according to the inference rules given in Fig. 6. We require that all I-tags occurring in D_T belong to IT_T and that all U-tags occurring in D_T belong to UT_T.

In Fig. 6, A ranges over *type environments* whose bindings $[z \mapsto u]$ map variables to union types. The inference rules involve *type judgements* of the form $A \vdash e : u$, which indicates that expression e has union type u in type environment A. All the type rules are "purely structural" in the sense that there is exactly one rule matching each syntactic form in $\mathcal{L}$, and it is defined in terms of type judgements on immediate subforms of the form. In particular, there is not a separate "subsumption rule" for subtyping. Instead, all subtyping relations have been "inlined" into the structural rules. This inlining simplifies the type/flow correspondence. Note that the typing of an expression ue^l does not depend on l, which may therefore be omitted.

The rules for intermediate configurations employ a predicate $se \bowtie A$, pronounced "*se is consistent with A*", that is defined as follows:

$$se \bowtie A \text{ iff } \forall z \in \mathit{dom}(se).\, \epsilon \vdash se(z) : A(z).$$

The use of the empty type environment within the definition of $\bowtie$ is sensible because a well-formed semantic environment se is closed. Because well-formed `bind` expressions are closed, the type environment A' used to analyze the e component depends only on se and is independent of the type environment A used in the conclusion of the [bind] rule. Similar commends hold in the [clos] rule.

The rules for function abstraction and function application are both instrumented with a "witness" that enables reconstructing the justification for applying the rule. In the application rule $[\mathsf{app}]^{w^{@}}$, the type of the operator e_1 is a (possibly empty) union type u_i, all of whose components have the expected function type $u_2 \to u$, but whose components' I-tagsets may differ. The app-*witness* $w^{@}$ is a function that maps each q in $dom(u_1)$ to an I-tagset that is appropriate for a particular instantiation of the application rule. Note that the app-witness in one instantiation of the application rule is independent of those chosen at any other instantiations of the application rule – a fact which is later critical for encoding polyvariant flow analyses in $\mathcal{TS}$. Intuitively, each instantiation of the application rule represents one of the possibly many contexts in which a polyvariant function is analyzed.

In the abstraction rule $[\mathsf{fun}]^{w^{\lambda}}$, the function types resulting from analyzing the function body in several different environments (intuitively, one per abstract application context) are combined into an intersection type t. This is wrapped into a union type with an arbitrary U-tag q, which provides a way of keeping track of the origin of a function type (cf. Sects. 1 and 5). Accordingly, the fun-*witness* w^{λ} of the abstraction rule is the pair $(q : t)$.

Remark 3.5. In $[\mathsf{fun}]^{w^{\lambda}}$, note that K may be empty, in which case the function body is not analyzed. This is an unusual feature of our type system that distinguishes it from most standard type systems for function-oriented languages. In typical type systems, a type derivation for an expression must include type derivations for all subexpressions. In our system, a function that is never applied (i.e., one which is "dead code") need not have a body that is well-typed. This allows flexibility for encoding in $\mathcal{TS}$ a wider range of flow analyses (many of which do not analyze dead code) than would be possible if type derivations were required for all function bodies. In the terminology of P&P, we let flows "have it their way". In contrast, P&P have chosen to let types "have it their way". They require that there is at least one type derivation for every function body, and consequently only consider flow analyses that analyze the bodies of every abstract closure. □

Example 3.6. For the program $\mathsf{P}_{2.1}$ from Ex. 2.1, we can construct the typing $\mathsf{T}_{2.1}$ shown in Fig. 7. Note that $u_{\mathtt{x}} \leq_u u'_{\mathtt{x}}$, and that $u_{\mathtt{x}}.q_{\mathtt{x}} \leq_t \bigwedge(\{2\} : u'_{\mathtt{x}} \to u'_{\mathtt{x}})$ so that $\{q_{\mathtt{x}} \mapsto \{2\}\}$ is indeed an app-witness for the inference at the top left of $D_{\lambda\mathtt{g}}$. □

Example 3.7. For the program $\mathsf{P}_{2.2}$ from Ex. 2.2, we can construct the typing $\mathsf{T}_{2.2}$ shown in Fig. 8. The derivation $D_{\mathsf{T}_{2.1}}$ in Fig. 8 demonstrates that we have the judgements

$$\epsilon \vdash \lambda^6\mathtt{g}.\mathtt{succ}^{13}\left(\left(\mathtt{g}^3 @_2\, \mathtt{g}^4\right) @_1\, 0^5\right) \,:\, \bigvee(q_{\mathtt{g}} : \bigwedge(\{\mathtt{x}\} : u_{\mathtt{x}} \to u_{\text{int}}, \{\mathtt{y}\} : u_{\mathtt{y}} \to u_{\text{int}}))$$
and
$$\epsilon \vdash \mathtt{if0}^9\; e_c\; \mathtt{then}\; \left(\lambda^8\mathtt{x}.\mathtt{x}^7\right)\; \mathtt{else}\; \left(\lambda^{12}\mathtt{y}.\lambda^{11}\mathtt{z}.\mathtt{z}^{10}\right) \,:\, u_{\mathtt{xy}}$$

which form a valid set of premises for $[\mathsf{app}]^{\{q_{\mathtt{g}} \mapsto \{\mathtt{x},\mathtt{y}\}\}}$ since

$$\bigwedge(\{\mathtt{x}\} : u_{\mathtt{x}} \to u_{\text{int}}, \{\mathtt{y}\} : u_{\mathtt{y}} \to u_{\text{int}}) \leq_t \bigwedge(\{\mathtt{x},\mathtt{y}\} : u_{\mathtt{xy}} \to u_{\text{int}}).$$

Note that the types are all finite, that $u_{\mathtt{y}} \leq_u u'_{\mathtt{y}}$, and that for all $q \in dom(u_{\mathtt{xy}})$ it holds that either $u_{\mathtt{xy}}.q = u_{\mathtt{x}}.q$ or $u_{\mathtt{xy}}.q = u_{\mathtt{y}}.q$.

This example also illustrates that the I-tags of a function reaching a call site need not match the sink label at that call site. For example, the app-*witness* at application site 0 is $\{q_{\mathtt{g}} \mapsto \{\mathtt{x},\mathtt{y}\}\}$ and the app-*witness* at application site 2 within $D_{\lambda\mathtt{g}_y}$ is $\{q_{\mathtt{x}} \mapsto \{3\}\}$. The app-*witness* effectively allows each copy of an application term in a derivation to have its own abstract sink labels that are independent of those from any other copy. □

3.4 Semantic soundness

The type system in Fig. 6 satisfies a subject reduction property, proved in Appendix A, which on top-level reads: If $\epsilon \vdash E \Rightarrow E'$ and $\epsilon \vdash E : u$ then also $\epsilon \vdash E' : u$:

$\mathsf{T}_{2.1} = \langle \mathsf{P}_{2.1}, IT_{\mathsf{T}_{2.1}}, UT_{\mathsf{T}_{2.1}}, D_{\mathsf{T}_{2.1}} \rangle$, where:

$$\begin{array}{rcl}
\mathsf{P}_{2.1} & = & \left(\lambda^6 \mathtt{g}.\left(\left(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4\right) \mathbin{@_1} 0^5\right)\right) \mathbin{@_0} \left(\lambda^8 \mathtt{x}.\mathtt{x}^7\right) \\
IT_{\mathsf{T}_{2.1}} & = & \{0,1,2\} \\
UT_{\mathsf{T}_{2.1}} & = & \left\{q_{\mathtt{x}}, q_{\mathtt{g}}\right\}
\end{array}$$

$D_{\mathsf{T}_{2.1}}$ is the type derivation tree:

$$\dfrac{D_{\lambda\mathtt{g}} \qquad\qquad D_{\lambda\mathtt{x}}}{\epsilon \vdash \left(\lambda^6 \mathtt{g}.\left(\left(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4\right) \mathbin{@_1} 0^5\right)\right) \mathbin{@_0} \left(\lambda^8 \mathtt{x}.\mathtt{x}^7\right) \,:\, u_{\mathrm{int}}}\;[\mathsf{app}]^{\{q_{\mathtt{g}} \mapsto \{0\}\}}$$

$D_{\lambda\mathtt{g}}$ is the type derivation tree:

$$\dfrac{\dfrac{\dfrac{A_{\mathtt{g}} \vdash \mathtt{g}^3 \,:\, u_{\mathtt{x}} \quad A_{\mathtt{g}} \vdash \mathtt{g}^4 \,:\, u'_{\mathtt{x}}}{A_{\mathtt{g}} \vdash \mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4 \,:\, u'_{\mathtt{x}}}\;[\mathsf{app}]^{\{q_{\mathtt{x}} \mapsto \{2\}\}} \qquad A_{\mathtt{g}} \vdash 0^5 \,:\, u_{\mathrm{int}}}{A_{\mathtt{g}} \vdash \left(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4\right) \mathbin{@_1} 0^5 \,:\, u_{\mathrm{int}}}\;[\mathsf{app}]^{\{q_{\mathtt{x}} \mapsto \{1\}\}}}{\epsilon \vdash \lambda^6 \mathtt{g}.\left(\left(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4\right) \mathbin{@_1} 0^5\right) \,:\, u_{\mathtt{g}}}\;[\mathsf{fun}]^{(q_{\mathtt{g}} : t_{\mathtt{g}})}$$

$D_{\lambda\mathtt{x}}$ is the type derivation tree:

$$\dfrac{A_{\mathtt{x}} \vdash \mathtt{x}^7 \,:\, u_{\mathrm{int}} \quad A'_{\mathtt{x}} \vdash \mathtt{x}^7 \,:\, u'_{\mathtt{x}}}{\epsilon \vdash \lambda^8 \mathtt{x}.\mathtt{x}^7 \,:\, u_{\mathtt{x}}}\;[\mathsf{fun}]^{(q_{\mathtt{x}} : t_{\mathtt{x}})}$$

$$\begin{array}{rcl}
u'_{\mathtt{x}} & = & \bigvee(q_{\mathtt{x}} : \bigwedge(\{1\} : u_{\mathrm{int}} \to u_{\mathrm{int}})) \\
t_{\mathtt{x}} & = & \bigwedge(\{1\} : u_{\mathrm{int}} \to u_{\mathrm{int}}, \{2\} : u'_{\mathtt{x}} \to u'_{\mathtt{x}}) \\
u_{\mathtt{x}} & = & \bigvee(q_{\mathtt{x}} : t_{\mathtt{x}}) \\
t_{\mathtt{g}} & = & \bigwedge(\{0\} : u_{\mathtt{x}} \to u_{\mathrm{int}}) \\
u_{\mathtt{g}} & = & \bigvee(q_{\mathtt{g}} : t_{\mathtt{g}}) \\
A'_{\mathtt{x}} & = & [\mathtt{x} \mapsto u'_{\mathtt{x}}] \\
A_{\mathtt{x}} & = & [\mathtt{x} \mapsto u_{\mathrm{int}}] \\
A_{\mathtt{g}} & = & [\mathtt{g} \mapsto u_{\mathtt{x}}]
\end{array}$$

Figure 7: A typing $T_{\mathsf{T}_{2.1}}$ for the program $\mathsf{P}_{2.1}$ from Ex. 2.1.

Theorem 3.8. *Suppose that with $se \bowtie A$ it holds that $se \vdash e \Rightarrow e'$ and $A \vdash e : u$. Then $A \vdash e' : u$.* □

As a consequence, "well-typed programs do not go wrong" as the following argument sketch demonstrates. For assume (in order to arrive at a contradiction) that $\epsilon \vdash P : u$ and that $\epsilon \vdash P \Rightarrow E$, where E is "stuck" in that E is not a value and yet for no E' it holds that $\epsilon \vdash E \Rightarrow E'$. By (repeated applications of) Theorem 3.8 we infer that $\epsilon \vdash E : u$. A case analysis reveals that within an "evaluation context" of E there exists an expression e that is of the form either

$$c \mathbin{@_l} sv \text{ or } \mathtt{succ}^l \left(\mathtt{close}^{l'}\; fn \;\mathtt{in}\; se\right) \text{ or } \mathtt{if0}^l \left(\mathtt{close}^{l'}\; fn \;\mathtt{in}\; se\right) \;\mathtt{then}\; e_1 \;\mathtt{else}\; e_2.$$

Since E contains e at an evaluation context, also e is typeable (that is, there exists A and u' such that $A \vdash e : u'$). But this is clearly impossible[7], and as desired we have arrived at a contradiction.

[7] Suppose that say $c \mathbin{@_l} sv$ is typeable, with the left premise taking the form $A \vdash c : u$. The side condition for [con] tells us that $q_{\mathrm{int}} \in dom(u)$, but this conflicts with the side condition for [app].

$\mathsf{T}_{2.2} = \langle \mathsf{P}_{2.2}, IT_{\mathsf{T}_{2.2}}, UT_{\mathsf{T}_{2.2}}, D_{\mathsf{T}_{2.2}} \rangle$, where:

$$\begin{array}{rcl}
\mathsf{P}_{2.2} & = & e_{\lambda\mathtt{g}} @_0\, e_{\mathrm{if}} \\
e_{\lambda\mathtt{g}} & = & \lambda^6\mathtt{g}.\mathtt{succ}^{13}\left(\left(\mathtt{g}^3 @_2\, \mathtt{g}^4\right) @_1\, 0^5\right) \\
e_{\mathrm{if}} & = & \mathtt{if0}^9\ e_c\ \mathtt{then}\ \left(\lambda^8\mathtt{x}.\mathtt{x}^7\right)\ \mathtt{else}\ \left(\lambda^{12}\mathtt{y}.\lambda^{11}\mathtt{z}.\mathtt{z}^{10}\right) \\
IT_{\mathsf{T}_{2.2}} & = & \{1, 2, 3, \mathtt{x}, \mathtt{y}\} \\
UT_{\mathsf{T}_{2.2}} & = & \left\{q_{\mathtt{x}}, q_{\mathtt{y}}, q_{\mathtt{z}}, q_{\mathtt{g}}\right\}
\end{array}$$

$D_{\mathsf{T}_{2.2}}$ is the type derivation tree:

$$\dfrac{\dfrac{D_{\lambda\mathtt{g}_x} \quad D_{\lambda\mathtt{g}_y}}{\epsilon \vdash e_{\lambda\mathtt{g}} : u_{\mathtt{g}}}\,[\mathsf{fun}]^{(q_{\mathtt{g}}:t_{\mathtt{g}})} \qquad D_{\mathrm{if}}}{\epsilon \vdash e_{\lambda\mathtt{g}} @_0\, e_{\mathrm{if}} : u_{\mathrm{int}}}\,[\mathsf{app}]^{\{q_{\mathtt{g}} \mapsto \{\mathtt{x},\mathtt{y}\}\}}$$

$D_{\lambda\mathtt{g}_x}$ is the type derivation tree:

$$\dfrac{\dfrac{\cdots (\mathit{Ex.\ 3.6}) \cdots}{[\mathtt{g} \mapsto u_{\mathtt{x}}] \vdash \left(\mathtt{g}^3 @_2\, \mathtt{g}^4\right) @_1\, 0^5 : u_{\mathrm{int}}}\,[\mathsf{app}]^{\{q_{\mathtt{x}} \mapsto \{1\}\}}}{[\mathtt{g} \mapsto u_{\mathtt{x}}] \vdash \mathtt{succ}^{13}\left(\left(\mathtt{g}^3 @_2\, \mathtt{g}^4\right) @_1\, 0^5\right) : u_{\mathrm{int}}}$$

$D_{\lambda\mathtt{g}_y}$ is the type derivation tree:

$$\dfrac{\dfrac{\dfrac{[\mathtt{g} \mapsto u_{\mathtt{y}}] \vdash \mathtt{g}^3 : u_{\mathtt{y}} \quad [\mathtt{g} \mapsto u_{\mathtt{y}}] \vdash \mathtt{g}^4 : u'_{\mathtt{y}}}{[\mathtt{g} \mapsto u_{\mathtt{y}}] \vdash \mathtt{g}^3 @_2\, \mathtt{g}^4 : u_{\mathtt{z}}}\,[\mathsf{app}]^{\{q_{\mathtt{x}} \mapsto \{3\}\}} \qquad [\mathtt{g} \mapsto u_{\mathtt{y}}] \vdash 0 : u_{\mathrm{int}}}{[\mathtt{g} \mapsto u_{\mathtt{y}}] \vdash \left(\mathtt{g}^3 @_2\, \mathtt{g}^4\right) @_1\, 0^5 : u_{\mathrm{int}}}\,[\mathsf{app}]^{\{q_{\mathtt{z}} \mapsto \{1\}\}}}{[\mathtt{g} \mapsto u_{\mathtt{y}}] \vdash \mathtt{succ}^{13}\left(\left(\mathtt{g}^3 @_2\, \mathtt{g}^4\right) @_1\, 0^5\right) : u_{\mathrm{int}}}$$

D_{if} is the type derivation tree:

$$\dfrac{\dfrac{\cdots (\mathit{Ex.\ 3.6}) \cdots}{\epsilon \vdash \lambda^8\mathtt{x}.\mathtt{x}^7 : u_{\mathtt{x}}}\,[\mathsf{fun}]^{(q_{\mathtt{x}}:t_{\mathtt{x}})} \qquad \dfrac{\dfrac{[\mathtt{y} \mapsto u'_{\mathtt{y}}][\mathtt{z} \mapsto u_{\mathrm{int}}] \vdash \mathtt{z} : u_{\mathrm{int}}}{[\mathtt{y} \mapsto u'_{\mathtt{y}}] \vdash \lambda^{11}\mathtt{z}.\mathtt{z}^{10} : u_{\mathtt{z}}}\,[\mathsf{fun}]^{(q_{\mathtt{z}}:t_{\mathtt{z}})}}{\epsilon \vdash \lambda^{12}\mathtt{y}.\lambda^{11}\mathtt{z}.\mathtt{z}^{10} : u_{\mathtt{y}}}\,[\mathsf{fun}]^{(q_{\mathtt{y}}:t_{\mathtt{y}})}}{\epsilon \vdash \mathtt{if0}^9\ e_c\ \mathtt{then}\ \left(\lambda^8\mathtt{x}.\mathtt{x}^7\right)\ \mathtt{else}\ \left(\lambda^{12}\mathtt{y}.\lambda^{11}\mathtt{z}.\mathtt{z}^{10}\right) : u_{\mathtt{xy}}}$$

$$\begin{array}{rcl}
u_{\mathtt{g}} & = & \bigvee(q_{\mathtt{g}} : \bigwedge(\{\mathtt{x}\} : u_{\mathtt{x}} \to u_{\mathrm{int}}, \{\mathtt{y}\} : u_{\mathtt{y}} \to u_{\mathrm{int}})) \\
u'_{\mathtt{x}} & = & \bigvee(q_{\mathtt{x}} : \bigwedge(\{1\} : u_{\mathrm{int}} \to u_{\mathrm{int}})) \\
t_{\mathtt{x}} & = & \bigwedge(\{1\} : u_{\mathrm{int}} \to u_{\mathrm{int}}, \{2\} : u'_{\mathtt{x}} \to u'_{\mathtt{x}}) \\
u_{\mathtt{x}} & = & \bigvee(q_{\mathtt{x}} : t_{\mathtt{x}}) \\
t_{\mathtt{z}} & = & \bigwedge(\{1\} : u_{\mathrm{int}} \to u_{\mathrm{int}}) \\
u_{\mathtt{z}} & = & \bigvee(q_{\mathtt{z}} : t_{\mathtt{z}}) \\
u'_{\mathtt{y}} & = & \bigvee(q_{\mathtt{y}} : \bigwedge()) \\
t_{\mathtt{y}} & = & \bigwedge(\{3\} : u'_{\mathtt{y}} \to u_{\mathtt{z}}) \\
u_{\mathtt{y}} & = & \bigvee(q_{\mathtt{y}} : t_{\mathtt{y}}) \\
u_{\mathtt{xy}} & = & \bigvee(q_{\mathtt{x}} : t_{\mathtt{x}}, q_{\mathtt{y}} : t_{\mathtt{y}})
\end{array}$$

Figure 8: Typing the program $\mathsf{P}_{2.2}$ from Example 2.2.

3.5 Embedding the Amadio & Cardelli Type System

In this section we investigate the relationship to the Amadio & Cardelli (AC) type system [AC93]. We show (Prop. 3.10) that AC can be embedded into the Mono part of our system, where a typing T belongs to Mono if it

1. is monovariant, that is IT_T is a singleton $\{\bullet\}$

$$
\begin{array}{lll}
[\mathsf{var}]^{AC} & B \vdash z^l : s & \text{if } B(z) \leq^{AC} s \\
[\mathsf{fun}]^{AC} & \dfrac{B[f \mapsto s_0][x \mapsto s_1] \vdash e : s_2}{B \vdash \mu f.\lambda^l x.e : s} & \text{if } s_1 \to s_2 \leq^{AC} s_0 \text{ and } s_1 \to s_2 \leq^{AC} s \\
[\mathsf{app}]^{AC} & \dfrac{B \vdash e_1 : s_1 \quad B \vdash e_2 : s_2}{B \vdash e_1 @_l e_2 : s} & \text{if } s_1 \leq^{AC} s_2 \to s
\end{array}
$$

Figure 9: The Amadio & Cardelli type system, adapted to a core subset of $\mathcal{L}$.

2. analyzes all code, that is all I-tagsets are non-empty.

In Mono, all intersection types are thus of the form $\bigwedge(\{\bullet\} : u_1 \to u_2)$ which we for the sake of brevity shall write as $u_1 \to u_2$.

Remark 3.9. The embedding proved directly in this section corresponds nicely to results proved later, since informally we can reason as follows: given a program P accepted by the Amadio & Cardelli type system, we know from [PO95] that P is also accepted by a 0-CFA that *(i)* has safety checks, and *(ii)* analyzes all subexpressions. The translation in Sec. 6 then shows that P can indeed be typed in Mono. □

We now briefly describe the AC type system. An AC type $s \in \mathcal{S}$ is a regular (potentially infinite) tree where each node is labelled by either

- `int`
- $\bot$
- $\top$
- $\to$

where the first three constructors are nullary, and the last one is binary. (One can think of $\bot$ as the empty union, and of $\top$ as a very big union.)

AC types can be equipped[8] with a partial order $\leq^{AC}$ with the following properties:

- $\forall s \in \mathcal{S}.\ \bot \leq^{AC} s$
- $\forall s \in \mathcal{S}.\ s \leq^{AC} \top$
- $\forall s_1, s_2, s_1', s_2' \in \mathcal{S}.\ s_1 \to s_2 \leq^{AC} s_1' \to s_2'$ iff $s_1' \leq^{AC} s_1$ and $s_2 \leq^{AC} s_2'$
- $\forall s_1, s_2 \in \mathcal{S}.\ s_1 \to s_2$ and `int` are incompatible wrt. $\leq^{AC}$.

For the sake of brevity, we shall only consider variables, function abstractions, and function applications (but the other constructs pose no additional problems). The corresponding inference rules for the AC type system are depicted in Fig. 9.

Proposition 3.10. *Suppose that we have a typing for P in* AC*. Then we can construct a* Mono *typing T for P.* □

The basic idea of our construction is to define UT_T as the least set containing `int` as well as all arrow types occurring (possibly deeply nested) within the AC typing. As AC types are regular, UT_T is a finite set. Then we can define a function

$\mathcal{T}^{AC}\colon \mathcal{S} \to \textbf{UnionType}$

[8]We shall be quite informal when it comes to reasoning about this ordering relation, but as in [KPS95] a rigorous treatment is possible, for instance using the technique of Sec. 3.2.

employing an auxiliary function $\mathcal{T}_0^{AC}\colon\ UT_T \to \mathbf{ElementaryType}$ by stipulating

$$\begin{array}{rcl}\mathcal{T}^{AC}(s) & = & \bigvee_{s' \in S} \{s' : \mathcal{T}_0^{AC}(s')\} \text{ where } S = \left\{s' \in UT_T \mid s' \leq^{AC} s\right\} \\ \mathcal{T}_0^{AC}(\mathtt{int}) & = & \mathtt{int} \\ \mathcal{T}_0^{AC}(s_1 \to s_2) & = & \mathcal{T}^{AC}(s_1) \to \mathcal{T}^{AC}(s_2)\end{array}$$

It is easy to see that for all s this uniquely defines a tree $\mathcal{T}^{AC}(s)$, which moreover is regular and thus a type. Identifying `int` with q_{int}, we have

$$\mathcal{T}^{AC}(\mathtt{int}) = \bigvee(\mathtt{int} : \mathtt{int}) = u_{\text{int}}.$$

Moreover, $\mathcal{T}^{AC}$ has been designed so as to be a monotone function: if $s_1 \leq^{AC} s_2$ then for all $q \in dom(\mathcal{T}^{AC}(s_1))$ we have $q \leq^{AC} s_1 \leq^{AC} s_2$ so $q \in dom(\mathcal{T}^{AC}(s_2))$ with $\mathcal{T}^{AC}(s_1).q = \mathcal{T}_0^{AC}(q) = \mathcal{T}^{AC}(s_2).q$ and therefore $\mathcal{T}^{AC}(s_1) \leq_u \mathcal{T}^{AC}(s_2)$. Also $\mathcal{T}_0^{AC}$ is monotone.

From the AC typing we now obtain a derivation D_T by replacing all judgements $B \vdash e : s$ by

$$\mathcal{T}^{AC}(B) \vdash e : \mathcal{T}^{AC}(s)$$

where $\mathcal{T}^{AC}$ extends pointwise to environments.

Prop. 3.10 now follows from the following Lemma, proved in App. A:

Lemma 3.11. *The tuple $\langle P, \{\bullet\}, UT_T, D_T\rangle$, as defined above, is a typing for P.* □

For the opposite direction, it is probably possible to translate a typing in Mono into a typing in AC, for example by stipulating

$$\begin{array}{rcl}\mathcal{T}_{AC}(\bigvee_{i\in I}\{q_i : t_i\}) & = & \sqcup_{i\in I}\mathcal{T}^0_{AC}(t_i) \\ \mathcal{T}^0_{AC}(\mathtt{int}) & = & \mathtt{int} \\ \mathcal{T}^0_{AC}(u_1 \to u_2) & = & \mathcal{T}_{AC}(u_1) \to \mathcal{T}_{AC}(u_2)\end{array}$$

where we employ that any finite set of AC types has a least upper bound (as well as a greatest lower bound). But we have not been able to convince ourselves that $\mathcal{T}_{AC}(u)$ is in general a regular tree.

3.6 Auxiliary Concepts

Addresses. In a typing T for P, for each e in $SubExpr_P$ there may be several judgements for e in D_T, due to the multiple analyses performed by [fun]. We assign to each judgement J for e in D_T an environment ke (its *address*) that for all applications of [fun] in the path from the root of D_T to J associates the bound variables with the branch taken.

Example 3.12. In $D_{\mathsf{T}_{2.1}}$ (Fig. 7), the judgement $A_{\mathtt{x}} \vdash \mathtt{x}^7 : u_{\text{int}}$ has address $[\mathtt{x} \mapsto 1]$ and the judgement $A'_{\mathtt{x}} \vdash \mathtt{x}^7 : u'_{\mathtt{x}}$ has address $[\mathtt{x} \mapsto 2]$. □

To be more formal: the root of a derivation D_T has address ϵ; if $A \vdash \mu f.\lambda^l x.e : u$ has address ke then the premise indexed by k has address $ke[f, x \mapsto k]$; and if J has address ke and is derived by something else than [fun] then all its premises have address ke.

Uniformity. The translation in Sect. 5 requires that a typing must be *uniform*, i.e., the following partial function A_T must be well-defined: $A_T(z,k) = u$ iff D_T contains a judgement of the form $A \vdash e : u'$ with address ke, where $ke(z) = k$ and $A(z) = u$.

Example 3.13. For $\mathsf{T}_{2.1}$ we have, e.g., $A_{\mathsf{T}_{2.1}}(\mathtt{x}, 1) = u_{\text{int}}$ and $A_{\mathsf{T}_{2.1}}(\mathtt{x}, 2) = u'_{\mathtt{x}}$. □

4 The Flow System

Our system for flow analysis has the form of a flow logic, in the style of N&N. A flow analysis F for program P is a tuple $\langle P, \mathit{Mem}_F, \mathcal{C}_F, \rho_F, \Phi_F\rangle$, where P is the program of interest and where the other components are explained below (together with some auxiliary concepts derivable from P and Mem_F).

Polyvariance is modeled by *mementoes*, where a memento ($m \in \mathit{Mem}_F$) represents a context for analyzing the body of a function. We shall assume that Mem_F is non-empty and *finite*; then all other entities occurring in F will also be finite. Each expression e is analyzed wrt. several different memento environments, where the entries of a memento environment ($me \in \mathit{MemEnv}_F$) take the form $[z \mapsto m]$ with m in Mem_F. Accordingly, a *flow configuration* ($\in \mathit{FlowConf}_F$) is a pair (e, me) ($e \in \mathit{LabExps}_{\mathrm{e}}(P)$ and $me \in \mathit{MemEnv}_F$), where $FV_{\mathrm{e}}(e) \subseteq \mathit{dom}(me)$.

The goal of the flow analysis is to associate a set of *flow values* to each configuration, where a flow value ($v \in \mathit{FlowVal}_F$) is either an integer Int or of the form (ac, M), where ac ($\in \mathit{AbsClos}_F$) is an *abstract closure* of the form (fn, me) with $fn \in \mathit{Funs}_{\mathrm{e}}(P)$ and $FV_{\mathrm{e}}(fn) \subseteq \mathit{dom}(me)$, and where $M \subseteq \mathit{Mem}_F$. The M component can be thought of as a superset of the "sinks" of the abstract closure ac, i.e., the contexts in which it is going to be applied.

In the design of flow values we deviate from N&N in two respects: *(i)* we do not include the memento that corresponds to the point of definition (as this is not relevant for our purposes); *(ii)* we do include the mementoes of use (the M component), in order to get a flow system that (as shown in Sect. 7) is almost isomorphic to the type system of Sect. 3. This extension does not make it harder to analyze an expression, since one might just let $M = \mathit{Mem}_F$ everywhere.

A *flow set* V ($\in \mathit{FlowSet}_F$) is a set of flow values, with the property that if $(ac, M_1) \in V$ and $(ac, M_2) \in V$ then $M_1 = M_2$. We define an ordering on $\mathit{FlowSet}_F$ by stipulating that $V_1 \leq_V V_2$ iff for all $v_1 \in V_1$ there exists $v_2 \in V_2$ such that $v_1 \leq_v v_2$, where the ordering $\leq_v$ on $\mathit{FlowVal}_F$ is defined by stipulating that $\mathrm{Int} \leq_v \mathrm{Int}$ and that $(ac, M_1) \leq_v (ac, M_2)$ iff $M_2 \subseteq M_1$. Note that if $V_1 \leq_V V_2$ then V_2 is obtained from V_1 by adding some "sources" and removing some "sinks" (in a sense moving along a "flow path" from a source to a sink), so in that respect the ordering is similar to the (shallow) type ordering in [WDMT97a]. It is easy to see that $\leq_V$ is reflexive and transitive, and that it makes $\mathit{FlowSet}_F$ a complete lattice. Note that $\mathrm{Int} \in \sqcap^V_{i\in I} V_i$ iff for all $i \in I$ it holds that $\mathrm{Int} \in I$; and that $(ac, M) \in \sqcap^V_{i\in I} V_i$ iff for all $i \in I$ there exists M_i such that $(ac, M_i) \in V_i$ with $M = \cup_{i\in I} M_i$. In particular, if $v \in \sqcap^V_{i\in I} V_i$ then for all $i \in I$ there exists $v_i \in V_i$ with $v \leq_v v_i$.

The function ϵ_v erases the M component from flow values so as to produce *unannotated flow values*, where an unannotated flow value ($uv \in \mathit{UnAnnFlowVal}_F$) is either an integer Int or an abstract closure; similarly the function ι_v produces flow values from unannotated flow values by annotating all abstract closures with Mem_F. That is, $\epsilon_v(\mathrm{Int}) = \mathrm{Int}$ and $\epsilon_v((ac, M)) = ac$ and $\iota_v(\mathrm{Int}) = \mathrm{Int}$ and $\iota_v(ac) = (ac, \mathit{Mem}_F)$; note that for all uv it holds that $\epsilon_v(\iota_v(uv)) = uv$. The functions ϵ_v and ι_v are trivially lifted to functions ϵ_V and ι_V between $\mathit{FlowSet}_F$ and $\mathcal{P}(\mathit{UnAnnFlowVal}_F)$. Note that if $V_1 \leq_V V_2$ then $\epsilon_V(V_1) \subseteq \epsilon_V(V_2)$.

Φ_F is a partial mapping from $(\mathit{Labs}_{\mathrm{e}}(P) \times \mathit{MemEnv}_F) \times \mathit{AbsClos}_F$ to $\mathcal{P}(\mathit{Mem}_F)$. Intuitively, if the abstract closure ac in the context me is applied to an expression with label l, then $\Phi_F((l, me), ac)$ denotes the actual sinks of ac.

$\mathcal{C}_F$ is a mapping from $\mathit{Labs}_{\mathrm{e}}(P) \times \mathit{MemEnv}_F$ to $(\mathit{FlowSet}_F)_\perp$. Intuitively, if $\mathcal{C}_F(l, me) = V$ ($\neq \perp$) and $\mathcal{C}_F$ is valid (defined below) for the flow configuration (ue^l, me) then all values that ue^l may evaluate to in a semantic environment approximated by me can be approximated by the set V. Similarly, $\rho_F(z, m)$ approximates the set of values to which z may be bound when analyzed in memento m.

Unlike N&N, we distinguish between $\mathcal{C}_F(l, me)$ being the empty set and being $\perp$. The latter means that no flow configuration (ue^l, me) is "reachable", and so there is no need to analyze it. The relation $\leq_V$ on $\mathit{FlowSet}_F$ is lifted to a relation $\leq_V$ on $\mathit{FlowSet}_{F\perp}$; note that $\perp \leq_V \emptyset$ is true whereas $\emptyset \leq_V \perp$ is false. $\mathit{FlowSet}_{F\perp}$ is a complete lattice, and $\sqcap^V_{i\in I} V_i = \perp$

iff there exists an $i \in I$ such that $V_i = \perp$. Also the predicate $\in$ can be lifted[9] in the natural way, so that, e.g., $v \notin \perp$ is considered a true statement.

EXAMPLE 4.1. For the program $\mathsf{P}_{2.1}$ from Ex. 2.1, a flow analysis $\mathsf{F}_{2.1}$ with $\mathit{Mem}_{\mathsf{F}_{2.1}} = \{0, 1, 2\}$ is given below. We have named some entities (note that $v_{\mathtt{x}} \leq_v v'_{\mathtt{x}}$):

$$\begin{array}{lll} me_{\mathtt{g}} = [\mathtt{g} \mapsto 0] & ac_{\mathtt{g}} = (\lambda\mathtt{g}.\cdots, \epsilon) & v_{\mathtt{g}} = (ac_{\mathtt{g}}, \{0\}) \\ me_{\mathtt{x1}} = [\mathtt{x} \mapsto 1] & ac_{\mathtt{x}} = (\lambda\mathtt{x}.\mathtt{x}^7, \epsilon) & v'_{\mathtt{x}} = (ac_{\mathtt{x}}, \{1\}) \\ me_{\mathtt{x2}} = [\mathtt{x} \mapsto 2] & & v_{\mathtt{x}} = (ac_{\mathtt{x}}, \{1, 2\}) \end{array}$$

$\mathcal{C}_{\mathsf{F}_{2.1}}$ and $\rho_{\mathsf{F}_{2.1}}$ are given by the entries below (all other are $\perp$):

$$\begin{array}{rcl} \{v_{\mathtt{g}}\} & = & \mathcal{C}_{\mathsf{F}_{2.1}}(6, \epsilon) \\ \{\mathrm{Int}\} & = & \rho_{\mathsf{F}_{2.1}}(\mathtt{x}, 1) = \mathcal{C}_{\mathsf{F}_{2.1}}(7, me_{\mathtt{x1}}) = \mathcal{C}_{\mathsf{F}_{2.1}}(5, me_{\mathtt{g}}) = \mathcal{C}_{\mathsf{F}_{2.1}}(1, me_{\mathtt{g}}) = \mathcal{C}_{\mathsf{F}_{2.1}}(0, \epsilon) \\ \{v'_{\mathtt{x}}\} & = & \rho_{\mathsf{F}_{2.1}}(\mathtt{x}, 2) = \mathcal{C}_{\mathsf{F}_{2.1}}(7, me_{\mathtt{x2}}) = \mathcal{C}_{\mathsf{F}_{2.1}}(4, me_{\mathtt{g}}) = \mathcal{C}_{\mathsf{F}_{2.1}}(2, me_{\mathtt{g}}) \\ \{v_{\mathtt{x}}\} & = & \rho_{\mathsf{F}_{2.1}}(\mathtt{g}, 0) = \mathcal{C}_{\mathsf{F}_{2.1}}(3, me_{\mathtt{g}}) = \mathcal{C}_{\mathsf{F}_{2.1}}(8, \epsilon) \end{array}$$

Thus $(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4) \mathbin{@_1} 0^5$ is analyzed with $\mathtt{g}$ bound to 0, and $\mathtt{x}^7$ is analyzed twice: with $\mathtt{x}$ bound to 1 and with $\mathtt{x}$ bound to 2. Accordingly, $\Phi_{\mathsf{F}_{2.1}}$ is given by

$$\Phi_{\mathsf{F}_{2.1}}((8, \epsilon), ac_{\mathtt{g}}) = \{0\},\ \Phi_{\mathsf{F}_{2.1}}((5, me_{\mathtt{g}}), ac_{\mathtt{x}}) = \{1\},\ \Phi_{\mathsf{F}_{2.1}}((4, me_{\mathtt{g}}), ac_{\mathtt{x}}) = \{2\}.$$

□

EXAMPLE 4.2. For the program $\mathsf{P}_{2.2}$ from Ex. 2.2, a flow analysis $\mathsf{F}_{2.2}$ with $\mathit{Mem}_{\mathsf{F}_{2.2}} = \{1, 2, 3, \mathtt{x}, \mathtt{y}\}$ is given below. We have named some entities:

$$\begin{array}{lll} me_{\mathtt{gx}} = [\mathtt{g} \mapsto \mathtt{x}] & ac_{\mathtt{g}} = (\lambda\mathtt{g}.\cdots, \epsilon) & v_{\mathtt{g}} = (ac_{\mathtt{g}}, \{\mathtt{x}, \mathtt{y}\}) \\ me_{\mathtt{gy}} = [\mathtt{g} \mapsto \mathtt{y}] & & v'_{\mathtt{x}} = (ac_{\mathtt{x}}, \{1\}) \\ me_{\mathtt{x1}} = [\mathtt{x} \mapsto 1] & ac_{\mathtt{x}} = (\lambda\mathtt{x}.\mathtt{x}^7, \epsilon) & v_{\mathtt{x}} = (ac_{\mathtt{x}}, \{1, 2\}) \\ me_{\mathtt{x2}} = [\mathtt{x} \mapsto 2] & & v'_{\mathtt{y}} = (ac_{\mathtt{y}}, \emptyset) \\ me_{\mathtt{y}} = [\mathtt{y} \mapsto 3] & ac_{\mathtt{y}} = (\lambda\mathtt{y}.\lambda^{11}\mathtt{z}.\mathtt{z}^{10}, \epsilon) & v_{\mathtt{y}} = (ac_{\mathtt{y}}, \{3\}) \\ me_{\mathtt{z}} = [\mathtt{y} \mapsto 3,\ \mathtt{z} \mapsto 1] & ac_{\mathtt{z}} = (\lambda\mathtt{z}.\mathtt{z}^{10}, me_{\mathtt{y}}) & v_{\mathtt{z}} = (ac_{\mathtt{z}}, \{1\}) \end{array}$$

$\mathcal{C}_{\mathsf{F}_{2.2}}$ and $\rho_{\mathsf{F}_{2.2}}$ are given by the entries below (all other are $\perp$):

$$\begin{array}{rcl} \{v_{\mathtt{g}}\} & = & \mathcal{C}_{\mathsf{F}_{2.2}}(6, \epsilon) \\ \{\mathrm{Int}\} & = & \rho_{\mathsf{F}_{2.2}}(\mathtt{x}, 1) = \rho_{\mathsf{F}_{2.2}}(\mathtt{z}, 1) = \mathcal{C}_{\mathsf{F}_{2.2}}(7, me_{\mathtt{x1}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(10, me_{\mathtt{z}}) \\ & = & \mathcal{C}_{\mathsf{F}_{2.2}}(5, me_{\mathtt{gx}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(5, me_{\mathtt{gy}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(1, me_{\mathtt{gx}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(1, me_{\mathtt{gy}}) \\ & = & \mathcal{C}_{\mathsf{F}_{2.2}}(13, me_{\mathtt{gx}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(13, me_{\mathtt{gy}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(0, \epsilon) \\ \{v'_{\mathtt{x}}\} & = & \rho_{\mathsf{F}_{2.2}}(\mathtt{x}, 2) = \mathcal{C}_{\mathsf{F}_{2.2}}(7, me_{\mathtt{x2}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(4, me_{\mathtt{gx}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(2, me_{\mathtt{gx}}) \\ \{v_{\mathtt{x}}\} & = & \rho_{\mathsf{F}_{2.2}}(\mathtt{g}, \mathtt{x}) = \mathcal{C}_{\mathsf{F}_{2.2}}(3, me_{\mathtt{gx}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(8, \epsilon) \\ \{v'_{\mathtt{y}}\} & = & \rho_{\mathsf{F}_{2.2}}(\mathtt{y}, 3) = \mathcal{C}_{\mathsf{F}_{2.2}}(4, me_{\mathtt{gy}}) \\ \{v_{\mathtt{y}}\} & = & \rho_{\mathsf{F}_{2.2}}(\mathtt{g}, \mathtt{y}) = \mathcal{C}_{\mathsf{F}_{2.2}}(3, me_{\mathtt{gy}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(12, \epsilon) \\ \{v_{\mathtt{z}}\} & = & \mathcal{C}_{\mathsf{F}_{2.2}}(2, me_{\mathtt{gy}}) = \mathcal{C}_{\mathsf{F}_{2.2}}(11, me_{\mathtt{y}}) \\ \{v_{\mathtt{x}}, v_{\mathtt{y}}\} & = & \mathcal{C}_{\mathsf{F}_{2.2}}(9, \epsilon) \end{array}$$

We see that $(\mathtt{g}^3 \mathbin{@_2} \mathtt{g}^4) \mathbin{@_1} 0^5$ is analyzed twice: with $\mathtt{g}$ bound to $\mathtt{x}$, and with $\mathtt{g}$ bound to $\mathtt{y}$. And in fact, $\Phi_{\mathsf{F}_{2.2}}$ is given by

$$\begin{array}{l} \Phi_{\mathsf{F}_{2.2}}((9, \epsilon), ac_{\mathtt{g}}) = \{\mathtt{x}, \mathtt{y}\} \\ \Phi_{\mathsf{F}_{2.2}}((5, me_{\mathtt{gx}}), ac_{\mathtt{x}}) = \{1\},\ \Phi_{\mathsf{F}_{2.2}}((5, me_{\mathtt{gy}}), ac_{\mathtt{z}}) = \{1\} \\ \Phi_{\mathsf{F}_{2.2}}((4, me_{\mathtt{gx}}), ac_{\mathtt{x}}) = \{2\},\ \Phi_{\mathsf{F}_{2.2}}((4, me_{\mathtt{gy}}), ac_{\mathtt{y}}) = \{3\} \end{array}$$

□

[9] We do *not* apply similar conventions for partial functions: for say $\Phi_F((l_2, me), ac) \subseteq M$ to be true, the left hand side must be defined.

$$
\begin{array}{ll}
[\mathit{var}] & F \models^{me} z^l \text{ iff } \bot \neq \rho_F(z, me(z)) \ \leq_V \ \mathcal{C}_F(l, me) \\
[\mathit{fun}] & F \models^{me} \mu f.\lambda^l x.e_0 \text{ iff } \{((\mu f.\lambda x.e_0, me), \mathit{Mem}_F)\} \leq_V \mathcal{C}_F(l, me) \\
[\mathit{app}] & F \models^{me} ue_1^{l_1} \ @_l \ ue_2^{l_2} \text{ iff} \\
& \qquad \mathcal{C}_F(l, me) \neq \bot \text{ and } F \models^{me} ue_1^{l_1} \text{ and } F \models^{me} ue_2^{l_2} \text{ and} \\
& \qquad \forall (ac_0, M_0) \in \mathcal{C}_F(l_1, me) \\
& \qquad\qquad \text{let } M = \Phi_F((l_2, me), ac_0) \text{ and } (\mu f.\lambda x.ue_0^{l_0}, me_0) = ac_0 \text{ in} \\
& \qquad\qquad M \subseteq M_0 \text{ and } \forall v \in \mathcal{C}_F(l_2, me).\, \exists m \in M.\, \{v\} \leq_V \rho_F(x, m) \text{ and} \\
& \qquad\qquad \forall m \in M\colon\ F \models^{me_0[f, x \mapsto m]} ue_0^{l_0} \text{ and} \\
& \qquad\qquad\qquad \mathcal{C}_F(l_0, me_0[f, x \mapsto m]) \leq_V \mathcal{C}_F(l, me) \text{ and} \\
& \qquad\qquad\qquad \rho_F(x, m) \neq \bot \text{ and } \{(ac_0, \mathit{Mem}_F)\} \leq_V \rho_F(f, m) \\
[\mathit{con}] & F \models^{me} c^l \text{ iff Int} \in \mathcal{C}_F(l, me) \\
[\mathit{suc}] & F \models^{me} \texttt{succ}^l\ e_1 \text{ iff } F \models^{me} e_1 \text{ and Int} \in \mathcal{C}_F(l, me) \\
[\mathit{if}] & F \models^{me} \texttt{if0}^l\ e_0 \ \texttt{then}\ ue_1^{l_1} \ \texttt{else}\ ue_2^{l_2} \text{ iff} \\
& \qquad F \models^{me} e_0 \text{ and } F \models^{me} ue_1^{l_1} \text{ and } F \models^{me} ue_2^{l_2} \text{ and} \\
& \qquad \mathcal{C}_F(l_1, me) \leq_V \mathcal{C}_F(l, me) \text{ and } \mathcal{C}_F(l_2, me) \leq_V \mathcal{C}_F(l, me) \\
[\mathit{bind}] & F \models^{me} \texttt{bind}^l\ se \ \texttt{in}\ ue_1^{l_1} \text{ iff} \\
& \qquad \exists me_1 \text{ with } se\ \mathcal{R}_F\ me_1\colon \\
& \qquad\qquad F \models^{me_1} ue_1^{l_1} \text{ and } \mathcal{C}_F(l_1, me_1) \leq_V \mathcal{C}_F(l, me) \\
[\mathit{clos}] & F \models^{me} \texttt{close}^l\ \mathit{fn} \ \texttt{in}\ se \text{ iff} \\
& \qquad \exists me_0 \text{ with } se\ \mathcal{R}_F\ me_0\colon\ \{((\mathit{fn}, me_0), \mathit{Mem}_F)\} \leq_V \mathcal{C}_F(l, me)
\end{array}
$$

Figure 10: The flow logic

4.1 Validity

Of course, not all flow analyses give a correct description of the program being analyzed. To formulate a notion of validity, we define a predicate $F \models^{me} e$ (to be read: F analyzes e correctly wrt. the memento environment me), with $(e, me) \in \mathit{FlowConf}_F$. The predicate must satisfy the specification in Fig. 10, where the clause for intermediate configurations employs a predicate $\mathcal{R}_F$ that is defined mutually recursively with another predicate $\mathcal{V}_F$:

$$
\begin{array}{rll}
se\ \mathcal{R}_F\ me & \text{iff} & \forall z \in \mathit{dom}(se)\colon\ se(z)\ \mathcal{V}_F\ \rho_F(z, me(z)) \\
c\ \mathcal{V}_F\ V & \text{iff} & \text{Int} \in V \\
(\texttt{close}\ \mathit{fn}\ \texttt{in}\ se)\ \mathcal{V}_F\ V & \text{iff} & \exists me \text{ with } se\ \mathcal{R}_F\ me\colon\ \{((\mathit{fn}, me), \mathit{Mem}_F)\} \leq_V V
\end{array}
$$

The specification in Fig. 10 gives rise to a monotone functional $\mathcal{G}_F$ on the complete lattice $\mathcal{P}(\mathit{FlowConf}_F)$; following the convincing argument of N&N, we define $F \models^{me} e$ as the *greatest* fixed point of this functional so as to be able to cope with recursive functions.

Concerning the rule [*fun*], we deviate from N&N by recording me, rather than the restriction of me to $FV_{\mathrm{e}}(\mu f.\lambda x.e_0)$. As in P&P, this facilitates the translations to and from types.

Concerning the rule [*app*], the set M corresponds to P&P's notion of *cover*, which in turn is needed to model the "cartesian product" algorithm of [Age95]. In N&N's framework, M is always a singleton $\{m\}$; in that case the condition "$\forall v \in \mathcal{C}_F(l_2, me).\ \ldots$ " amounts to the simpler "$\mathcal{C}_F(l_2, me) \leq_V \rho_F(x, m)$". On the other hand, unlike N&N we do not give freedom to introduce new mementoes elsewhere.

In examples involving non-recursive functions of the form $\lambda x.e$, this being a shorthand for $\mu f.\lambda x.e$ where f does not occur in e, we shall not bother about the set $\rho_F(f, m)$ and in particular implicitly discard the last clause in the rule $[app]$.

Note that even if e is a subexpression of a program P with $F \models^{\epsilon} P$ then it is not necessarily the case that $F \models^{me} e$ for some me; this might happen if e is "dead code". But keep in mind that we work under a "closed world" assumption: if $P = \lambda x.e$ then e is dead code!

By structural induction in ue^l we see that if $F \models^{me} ue^l$ then $\mathcal{C}_F(l, me) \neq \bot$. We would also like the converse implication to hold:

Definition 4.3. Let a flow analysis F for P be given. We say that F is valid iff *(i)* $F \models^{\epsilon} P$; *(ii)* whenever $e = ue^l \in \mathit{SubExpr}_P$ with $(e, me) \in \mathit{FlowConf}_F$ and $\mathcal{C}_F(l, me) \neq \bot$ then $F \models^{me} e$. □

4.2 Semantic Soundness.

Our flow logic satisfies a subject reduction property, proved in Appendix B.1 using techniques as in N&N, which for closed E reads: if $\epsilon \vdash E \Rightarrow E'$ and $F \models^{\epsilon} E$ then $F \models^{\epsilon} E'$:

Theorem 4.4. *Suppose that with* $se \ \mathcal{R}_F \ me$ *it holds that* $se \vdash e \Rightarrow e'$ *and* $F \models^{me} e$. *Then* $F \models^{me} e'$. □

As a consequence, we see that a flow analysis F indeed is a "closure analysis": if $se \ \mathcal{R}_F \ me$ and $se \vdash ue^l \Rightarrow^* \mu f.\lambda^l x.e_0$ and $F \models^{me} ue^l$ then $((\mu f.\lambda x.e_0, me), M) \in \mathcal{C}_F(l, me)$ for some M. For by (repeated applications of) Theorem 4.4 we infer $F \models^{me} \mu f.\lambda^l x.e_0$, that is $\{((\mu f.\lambda x.e_0, me), \mathit{Mem}_F)\} \leq_V \mathcal{C}_F(l, me)$. Note that this result hinges on the fact that $se \vdash ue_1^{l_1} \Rightarrow ue_2^{l_2}$ implies $l_1 = l_2$ (unlike what is the case in P&P).

So far, even for badly behaved programs like $P = 7 \,@\, 9$ it is possible (just as in N&N) to find a F for P such that F is valid. Since our type system rejects such programs, we would like to filter them out (in this respect "letting types have it their way"):

Definition 4.5. Let a flow analysis F for P be given. We say that F is safe iff for all ue^l in $\mathit{SubExpr}_P$ and for all me it holds: *(i)* if $ue = ue_1^{l_1} \,@\, e_2$ then $\mathrm{Int} \notin \mathcal{C}_F(l_1, me)$; *(ii)* if $ue = \mathtt{succ}\ ue_1^{l_1}$ then $v \in \mathcal{C}_F(l_1, me)$ implies $v = \mathrm{Int}$; *(iii)* if $ue = \mathtt{if0}\ ue_0^{l_0}\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2$ then $v \in \mathcal{C}_F(l_0, me)$ implies $v = \mathrm{Int}$. □

Example 4.6. Referring back to Examples 4.1 and 4.2, it clearly holds that $\mathsf{F}_{2.1}$ is safe and $\mathsf{F}_{2.2}$ is safe, and it is easy (though a little cumbersome) to verify that $\mathsf{F}_{2.1}$ is valid and $\mathsf{F}_{2.2}$ is valid. □

4.3 Taxonomy of Flow Analyses

Two common categories of flow analyses are the "call-string based" (e.g., [Shi91]) and the "argument-based" (e.g., [Sch95, Age95]). Below we shall see that our *descriptive* framework can model both approaches (which can be "mixed", as in [NN99]).

A flow analysis F for P such that F is valid is in $\mathit{CallString}_\beta^P$, where β is a mapping from $\mathit{Labs}_\mathrm{e}(P) \times \mathit{MemEnv}_F$ into Mem_F, iff whenever $\Phi_F((l_2, me), ac)$ is defined it equals $\{\beta(l, me)\}$ where l is such that[10] $e_1 \,@_l\, ue_2^{l_2} \in \mathit{SubExpr}_P$. All k-CFA analyses fit into this category: for 0-CFA we take $\mathit{Mem}_F = \{\bullet\}$ and $\beta(l, me) = \bullet$; for 1-CFA we take $\mathit{Mem}_F = \mathit{Labs}_\mathrm{e}(P)$ and $\beta(l, me) = l$; and for 2-CFA (the generalization to $k > 2$ is immediate) we take $\mathit{Mem}_F = \mathit{Labs}_\mathrm{e}(P) \cup (\mathit{Labs}_\mathrm{e}(P) \times \mathit{Labs}_\mathrm{e}(P))$ and define $\beta(l, me)$ as follows: let it be l if $me = \epsilon$, and let it be (l, l_1) if me takes the form $me'[z \mapsto m]$ with m either l_1 or (l_1, l_2).

Example 4.7. The flow analysis $\mathsf{F}_{2.1}$ is a 1-CFA, whereas the flow analysis $\mathsf{F}_{2.2}$ is not in $\mathit{CallString}_\beta^{\mathsf{P}_{2.2}}$ for any β (since $\Phi_{\mathsf{F}_{2.2}}$ in one case returns a doubleton). □

[10] It is tempting to write "$\Phi_F((l, me), ac_0)$" in Fig. 10 (thus replacing l_2 by l), but then subject reduction for flows would not hold.

A flow analysis F for P such that F is valid is in $\mathit{ArgBased}^P_\alpha$, where α is a total mapping from Mem_F into $\mathcal{P}(\mathit{UnAnnFlowVal}_F)$, iff for all λ-bound variables x and mementoes m it holds that whenever $\rho_F(x,m) \neq \perp$ then $\epsilon_V(\rho_F(x,m)) = \alpha(m)$. For this kind of analysis, a memento m essentially denotes a set of unannotated flow values. We may impose further demands on α so as to more precisely capture specific brands of argument-based analyses, such as [Age95] or the type-directed approach of [JWW97]; below we shall treat two interesting subcategories: if $\alpha(m_1)$ and $\alpha(m_2)$ are disjoint whenever $m_1 \neq m_2$ we write Disj_α; and if α satisfies that each element in $\mathit{UnAnnFlowVal}_F$ occurs in at least one $\alpha(m)$ we write Cover_α.

Example 4.8. The flow analysis $\mathsf{F}_{2.1}$ is in $\mathit{ArgBased}^{\mathsf{P}_{2.1}}_\alpha$, with $\alpha(0) = \alpha(2) = \{ac_{\mathtt{x}}\}$ and $\alpha(1) = \{\mathrm{Int}\}$. The flow analysis $\mathsf{F}_{2.2}$ is in $\mathit{ArgBased}^{\mathsf{P}_{2.2}}_\alpha$, with $\alpha(\mathtt{x}) = \alpha(2) = \{ac_{\mathtt{x}}\}$ and $\alpha(\mathtt{y}) = \alpha(3) = \left\{ac_{\mathtt{y}}\right\}$ and $\alpha(1) = \{\mathrm{Int}\}$.

Note that by appropriate renaming (collapsing) of mementoes, both flow analyses can be converted so as to fit into a class $\mathit{ArgBased}_\alpha$ with Disj_α. □

4.4 Existence of Least Analyses

Given a program P, it turns out that for all β the class $\mathit{CallString}^P_\beta$, and for certain kinds of α also the class $\mathit{ArgBased}^P_\alpha$, contains a least (i.e., most precise) flow analysis; here the ordering on flow analyses is defined pointwise[11] on $\mathcal{C}_F$, ρ_F and Φ_F. This is much as in N&N where for all total and deterministic "instantiators" the corresponding class of analyses contains a least element, something we cannot hope for since we allow Φ_F to return a non-singleton[12].

Theorem 4.9. *For all P and β the class $\mathit{CallString}^P_\beta$ contains a least flow analysis; and for all P and α with Disj_α the class $\mathit{ArgBased}^P_\alpha$ contains a least flow analysis provided it is not empty—a sufficient condition for which is that Cover_α.* □

Proof. The theorem is an immediate consequence of Lemmas B.5 and B.6, stated and proved in Appendix B.2. □

4.5 Encoding the P&P Framework

Our flow system was developed along the lines of N&N, generalizing some features (while omitting other). That the resulting framework has substantial descriptive power is indicated by the fact that the framework of P&P, designed so as to model several existing flow analyses, can be encoded into our framework—even though on the surface it is quite different from ours.

A P&P analysis of a program P is a set R of triples of the form (ρ, e, s) with $\rho \in \mathit{FlowEnv}(P)$ and e an expression $\in \mathit{LabExps}_{\mathrm{e}}(P)$ and $s \in \mathcal{P}(\mathit{Val}(P))$. Here $\mathit{FlowEnv}(P)$ and $\mathit{Val}(P)$ are recursively defined (note that values may be infinite structures):

- a value $a \in \mathit{Val}(P)$ is either Int or a pair $(\lambda x.e, \rho)$ with $\rho \in \mathit{FlowEnv}(P)$;
- an environment $\rho \in \mathit{FlowEnv}(P)$ has entries of the form $[x \mapsto s]$ with $s \in \mathcal{P}(\mathit{Val}(P))$.

As in P&P we say that an analysis of P is an F-analysis if *(i)* it satisfies certain criteria, listed in Fig. 11, ensuring that "the analysis gives a correct description of P"; and *(ii)* there exists s such that $(\epsilon, P, s) \in R$.

As values in P&P may be infinite whereas in our framework all flow values are finite structures, it seems hard to embed the former into the latter. This is when the "finitary" condition, which by the way is necessary for P&P's translation into types, comes to rescue:

[11] Unlike [JWW97], we do not compare analyses with different sets of mementoes: if F_1 and F_2 are two flow analyses for P we stipulate that $F_1 \leq_F F_2$ holds iff *(i)* $\mathit{Mem}_{F_1} = \mathit{Mem}_{F_2}$; *(ii)* $\mathcal{C}_{F_1}(l, me) \leq_V \mathcal{C}_{F_2}(l, me)$ for all l and me; *(iii)* $\rho_{F_1}(z,m) \leq_V \rho_{F_2}(z,m)$ for all z and m; *(iv)* $\Phi_{F_1}((l,me),ac) \subseteq \Phi_{F_2}((l,me),ac)$ whenever the left hand side is defined.

[12] The "culprit" is the condition for [*app*] "$\forall v \in \mathcal{C}_F(l_2, me).\, \exists m \in M.\, \{v\} \leq_V \rho_F(x,m)$".

$(\rho, x^l, s) \in R$ implies $\rho(x) \subseteq s$

$(\rho, \lambda^l x.e, s) \in R$ implies $(\lambda x.e, \rho) \in s$

$(\rho, e_1 @_l e_2, s) \in R$ implies $\exists s_1, s_2.$
$\quad (\rho, e_1, s_1) \in R$ and $(\rho, e_2, s_2) \in R$ and
$\quad \forall (\lambda x.e, \rho') \in s_1. \exists C.$
$\qquad s_2 \subseteq \cup C$ and
$\qquad \forall s' \in C. \exists s''.$
$\qquad\quad (\rho'[x \mapsto s'], e, s'') \in R$ and $s'' \subseteq s$

$(\rho, c^l, s) \in R$ implies $\mathrm{Int} \in s$

$(\rho, \mathtt{succ}^l\ e_1, s) \in R$ implies $\mathrm{Int} \in s$ and $\exists s_1.$
$\quad (\rho, e_1, s_1) \in R$

$(\rho, \mathtt{if0}^l\ e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2, s) \in R$ implies $\exists s_0, s_1, s_2.$
$\quad s_1 \subseteq s$ and $s_2 \subseteq s$ and for $i = 0, 1, 2$: $(\rho, e_i, s_i) \in R$

Figure 11: Specification of an F-analysis.

Definition 4.10. Let R be a P&P analysis. We say that R is *finitary* if there exists a finite set Val_{fin} of (possibly infinite) flow values such that all values occurring (possibly deeply nested) in R are members of Val_{fin}. To be more precise: with $FlowEnv_{fin}$ the environments in $FlowEnv(P)$ where all entries are of the form $[x \mapsto s]$ with $s \in \mathcal{P}(Val_{fin})$, it must hold that

- if $a \in Val_{fin}$ then a is either Int or a pair $(\lambda x.e, \rho)$ with $\rho \in FlowEnv_{fin}$;
- each element in R takes the form (ρ, e, s) with $\rho \in FlowEnv_{fin}$ and $s \in \mathcal{P}(Val_{fin})$.

□

Additionally, it seems that we must also assume that R has the following *deterministic cache property*: there exists a function $Cach_R$ such that in the case for application in Fig. 11 we can choose the "cache" C to be $Cach_R((e_1 @_l e_2, \rho), (\lambda x.e, \rho'))$ (that is, C does not depend on s, s_1, or s_2).

Proposition 4.11. *Suppose that R is a finitary F-analysis for P that has the deterministic cache property. Then we can from R construct a flow analysis F such that F is valid. Moreover, F belongs to $ArgBased^P_\alpha$ for some α.* □

The basic idea of our construction is—with Val_{fin} and $FlowEnv_{fin}$ as in Definition 4.10—to define Mem_F as a set of the same size of $\mathcal{P}(Val_{fin})$. Let γ be a bijection between $\mathcal{P}(Val_{fin})$ and Mem_F; this mapping in the obvious way induces a bijection γ_{me} from $FlowEnv_{fin}$ to $MemEnv_F$. We next extend γ_{me} to a mapping γ_v from Val_{fin} to $FlowVal_F$:

$$\begin{array}{rcl} \gamma_v(\mathrm{Int}) & = & \mathrm{Int} \\ \gamma_v((\lambda x.e, \rho)) & = & ((\lambda x.e, \gamma_{me}(\rho)), Mem_F) \end{array}$$

And in the obvious way we extend γ_v to a mapping γ_V from $\mathcal{P}(Val_{fin})$ to $FlowSet_F$.

We now define $\mathcal{C}_F$ and ρ_F, by stipulating

$$\rho_F(x, m) = \gamma_V(\gamma^{-1}(m))$$

and for $ue^l \in SubExpr_P$ stipulating

$$\begin{array}{l} \mathcal{C}_F(l, me) = \text{let } S = \{s \mid (\gamma_{me}^{-1}(me), ue^l, s) \in R\} \\ \qquad\qquad\quad \text{in if } S = \emptyset \text{ then } \bot \text{ else } \gamma_V(\cap S) \end{array}$$

We define Φ_F using Cach_R (cf. above): if $\mathit{Cach}_R((e_1 \,@_l\, ue_2^{l_2}, \rho), (\lambda x.e, \rho')) = C$ then $\Phi_F((l_2, \gamma_{me}(\rho)), (\lambda x.e, \gamma_{me}(\rho'))) = \{\gamma(s) \mid s \in C\}$.

This completes the definition of a flow analysis F. Proposition 4.11 now follows from the following lemma, proved in Appendix B.3, together with the observation that F belongs to $\mathit{ArgBased}_\alpha^P$ with $\alpha(m) = \epsilon_V(\gamma_V(\gamma^{-1}(m)))$.

Lemma 4.12. *Let a flow analysis F be defined as above. Then F is valid.* □

The framework of P&P can model 0-CFA, by requiring every cache C to be a singleton and to depend only on the function body. Such an analysis is by the construction depicted above translated into a flow analysis where $\Phi_F((l, me), (\lambda x.e, me_0))$ is a singleton depending on $\lambda x.e$ only. So for each z there exists only one m such that $\rho_F(z, m)$ is of interest. Therefore one might collapse all mementoes into a single memento, corresponding to the way 0-CFA is modeled in our framework (cf. Sect. 4.3).

One can also go the other direction, and convert an *argument-based* flow analysis into a finitary F-analysis:

Proposition 4.13. *Suppose that F is a flow analysis for P such that F is valid, and that $F \in \mathit{ArgBased}_\alpha^P$. Then we can from F construct a finitary F-analysis for P.* □

The idea is to use α: $\mathit{Mem}_F \to \mathcal{P}(\mathit{UnAnnFlowVal}_P)$ to write mutually recursive functions

δ_{me}: $\mathit{MemEnv}_F \to \mathit{FlowEnv}(P)$

δ_{uv}: $\mathit{UnAnnFlowVal}_F \to \mathit{Val}(P)$

δ_V: $\mathit{FlowSet}_F \to \mathcal{P}(\mathit{Val}(P))$

as follows:

$$\begin{aligned}
\delta_{me}(me)(x) &= \{\delta_{uv}(uv) \mid uv \in \alpha(me(x))\} \\
\delta_{uv}(\mathrm{Int}) &= \mathrm{Int} \\
\delta_{uv}((\lambda x.e, me)) &= (\lambda x.e, \delta_{me}(me)) \\
\delta_V(V) &= \{\delta_{uv}(uv) \mid uv \in \epsilon_V(V)\}
\end{aligned}$$

This is clearly well-defined (but the output may contain infinite structures). Note that δ_V is monotonic: if $V_1 \leq_V V_2$ then $\delta_V(V_1) \subseteq \delta_V(V_2)$. We then stipulate

$$R = \big\{(\delta_{me}(me), e, \delta_V(\mathcal{C}_F(l, me))) \mid e = ue^l \in \mathit{SubExpr}_P \text{ and } F \models^{me} e\big\}.$$

which is well-defined (as $\mathcal{C}_F(l, me) \neq \perp$ if $F \models^{me} e$) and clearly finitary.

Proposition 4.13 now follows from the following lemma, proved in Appendix B.3.

Lemma 4.14. *Let R be defined as above. Then R is an F-analysis.* □

The relationship between the two translations presented in this section, such as whether the roundtrips might be the identity, is left for future work.

4.6 Reachability

For a flow analysis F, some entries may be garbage. To see an example of this, suppose that $\mu f.\lambda x.ue^l$ in $\mathit{SubExpr}_P$, and suppose that $\rho_F(x, m) = \perp$ for all $m \in \mathit{Mem}_F$. From this we infer that the above function is never called, so for all me the value of $\mathcal{C}_F(l, me)$ is uninteresting. It may therefore be replaced by $\perp$, something which is in fact achieved by the roundtrip described in Sect. 7.1.

To formalize a notion of reachability we introduce a set Reach_P^F that is intended to encompass[13] all entries of $\mathcal{C}_F$ and ρ_F that are "reachable" from the root of P. Let $\mathit{Analyzes}_m^F(\mu f.\lambda x.ue_0^{l_0}, me)$

[13]This is somewhat similar to the reachability predicate of [GNN97].

be a shorthand for $\mathcal{C}_F(l_0, me[f,x \mapsto m]) \neq \bot$ and $\rho_F(x,m) \neq \bot$ and $\Big\{((\mu f.\lambda x.ue_0^{l_0}, me), Mem_F)\Big\} \leq_V \rho_F(f,m)$. We define $Reach_P^F$ as the least set satisfying:

[prg] $(P, \epsilon) \in Reach_P^F$

[fun] $\Big((\mu f.\lambda^l x.ue_0^{l_0}, me) \in Reach_P^F$ and $Analyzes_m^F(\mu f.\lambda x.ue_0^{l_0}, me)\Big) \Rightarrow$
$\Big\{(ue_0^{l_0}, me[f,x \mapsto m]),\ (x,m),\ (f,m)\Big\} \subseteq Reach_P^F$

[app] $(e_1 @_l e_2, me) \in Reach_P^F \Rightarrow \{(e_1, me),\ (e_2, me)\} \subseteq Reach_P^F$

[suc] $(\texttt{succ}^l\ e_1, me) \in Reach_P^F \Rightarrow (e_1, me) \in Reach_P^F$

[if] $(\texttt{if0}^l\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2, me) \in Reach_P^F \Rightarrow$
$\{(e_0, me),\ (e_1, me),\ (e_2, me)\} \subseteq Reach_P^F$

Example 4.15. It is easy to verify that for $ue^l \in SubExpr_{\mathsf{P}_{2.1}}$ it holds that $\mathcal{C}_{\mathsf{F}_{2.1}}(l, me) \neq \bot$ iff $(ue^l, me) \in Reach_{\mathsf{P}_{2.1}}^{\mathsf{F}_{2.1}}$, and that $\rho_{\mathsf{F}_{2.1}}(z,m) \neq \bot$ iff $(z,m) \in Reach_{\mathsf{P}_{2.1}}^{\mathsf{F}_{2.1}}$. Similarly for $\mathsf{P}_{2.2}$ and $\mathsf{F}_{2.2}$. □

Note that if $(e, me) \in Reach_P^F$ then $e \in SubExpr_P$ and $(e, me) \in FlowConf_F$. The following result, proved in Appendix B.4, shows that reachability implies definedness provided that the flow analysis is valid.

Lemma 4.16. *Let F be a flow analysis for P such that F is valid. If $(ue^l, me) \in Reach_P^F$ then* (i) *$\mathcal{C}_F(l, me) \neq \bot$ and* (ii) *whenever $(z \mapsto m) \in me$ then $(z,m) \in Reach_P^F$ holds. Also, if $(z,m) \in Reach_P^F$ then $\rho_F(z,m) \neq \bot$.* □

5 Translating Types to Flows

Let a uniform typing T for a program P be given. We now demonstrate how to construct a corresponding flow analysis $F = \mathcal{F}(T)$ such that F is valid and safe. First define Mem_F as IT_T; note that then an address can serve as a memento environment. Next we define a function $\mathcal{F}_T$ that translates from $UTyp_T$, that is the union types that can be built using IT_T and UT_T, into $FlowSet_F$:

$$\begin{array}{l}
\mathcal{F}_T(\bigvee_{i \in I}\{q_i : t_i\}) = \\
\quad \{((\mu f.\lambda x.e, me), M) \mid \exists i \in I \text{ with } M = dom(t_i): \\
\qquad \text{a judgement for } \mu f.\lambda^l x.e \text{ occurs in } D_T \text{ with address } me \\
\qquad\quad \text{and is justified by } [\mathsf{fun}]^{(q_i:t)} \text{ where } t \leq_t t_i\} \\
\quad \cup (\text{if } \exists i. \text{ such that } q_i = q_{\text{int}} \text{ then } \{\text{Int}\} \text{ else } \emptyset)
\end{array}$$

The idea behind the translation is that $\mathcal{F}_T(u)$ should contain all the closures that are "sources" of elementary types in u; it is easy to trace such closures thanks to the presence of U-tags. The condition $t \leq_t t_i$ is needed as a "sanity check", quite similar to the "trimming" performed in [Hei95], to guard against the possibility that two unrelated entities in D_T incidentally have used the same U-tag q_i. As the types of P&P do not contain fun-witnesses, their translation has to rely solely on this sanity check (at the cost of precision, cf. the example given in Sect. 1).

Example 5.1. With terminology as in Examples 3.6 and 4.1, it is easy to see that $\mathcal{F}_{\mathsf{T}_{2.1}}(u'_{\mathtt{x}}) = \{v'_{\mathtt{x}}\}$ and that $\mathcal{F}_{\mathsf{T}_{2.1}}(u_{\mathtt{x}}) = \{v_{\mathtt{x}}\}$. □

Lemma 5.2. *The function $\mathcal{F}_T$ is monotone.* □

Proof. Assume that $u \leq_u u'$. Let $v \in \mathcal{F}_T(u)$ be given; we must show that there exists $v' \in \mathcal{F}_T(u')$ such that $v \leq_v v'$. First assume that $v = \text{Int}$. Then $q_{\text{int}} \in dom(u)$, so also $q_{\text{int}} \in dom(u')$ implying $\text{Int} \in \mathcal{F}_T(u')$.

Next assume that v takes the form (ac, K), with $ac = (\mu f.\lambda x.e, me)$. There thus exists $q \in dom(u)$ such that $K = dom(u.q)$ and such that a judgement for $\mu f.\lambda x.e$ occurs in D_T with address me and is derived by $[\mathsf{fun}]^{(q:t)}$ where $t \leq_t u.q$. Since $u \leq_u u'$ we have $u.q \leq_t u'.q$, implying

$$t \leq_t u'.q \text{ and } dom(u'.q) \subseteq dom(u.q).$$

Let $K' = dom(u'.q)$ and $v' = (ac, K')$. From the above we infer the desired relations: $v' \in \mathcal{F}_T(u')$ where $v \leq_v v'$ (since $K' \subseteq K$). □

Lemma 5.3. $\mathcal{F}_T(u_{\text{int}}) = \{Int\}$. □

Definition 5.4. With T a typing for P, the flow analysis $F = \mathcal{F}(T)$ is given by $\langle P, IT_T, \mathcal{C}_F, \rho_F, \Phi_F \rangle$, where $\mathcal{C}_F$, ρ_F, and Φ_F are defined below:

$\mathcal{C}_F(l, me) = \mathcal{F}_T(u)$ iff D_T contains a judgement $A \vdash ue^l : u$ with address me
$\rho_F(z, m) = \mathcal{F}_T(u)$ iff $u = A_T(z, m)$
$\Phi_F((l_2, me), (\mu f.\lambda x.e_0, me')) = M$ iff there exists q such that D_T contains
a judgement for $\mu f.\lambda x.e_0$ at me' derived by $[\mathsf{fun}]^{(q:t)}$,
a judgement for $e_1 @ ue_2^{l_2}$ at me derived by $[\mathsf{app}]^{w^@}$ where $w^@(q) = M$.

□

Example 5.5. It is easy to check that $\mathsf{F}_{2.1} = \mathcal{F}(\mathsf{T}_{2.1})$, and that $\mathsf{F}_{2.2} = \mathcal{F}(\mathsf{T}_{2.2})$. □

Theorem 5.6. *With T a uniform typing for P, for $F = \mathcal{F}(T)$ it holds that*

- *F is valid and safe*
- *$(ue^l, me) \in Reach_P^F$ iff $\mathcal{C}_F(l, me) \neq \bot$ (for $ue \in SubExpr_P$)*
- *$(z, m) \in Reach_P^F$ iff $\rho_F(z, m) \neq \bot$*

□

Proof. The result follows from Lemmas C.2, C.3, C.4, C.5, and C.6 that are all established in Appendix C. In particular, note that the proof that F is valid is by coinduction. □

6 Translating Flows to Types

Let a flow analysis F for a program P be given, and assume that F is valid and safe. We now demonstrate how to construct a corresponding uniform typing $T = \mathcal{T}(F)$. First we define IT_T as Mem_F and UT_T as $AbsClos_F \cup \{q_{\text{int}}\}$. Next we define a function $\mathcal{T}_F$ that translates from $FlowSet_F$ into $UTyp_T$; inspired by P&P (though the setting is somewhat different) we stipulate:

$\mathcal{T}_F(V) = \bigvee_{v \in V} \{q_v : t_v\}$ where
if $v = \text{Int}$ then $q_v = q_{\text{int}}$ and $t_v = \texttt{int}$
if $v = (ac, M)$ with $ac = (\mu f.\lambda x.e_0^{l_0}, me)$
then $q_v = ac$
and $t_v = \bigwedge_{m \in M_0} \{\{m\} : \mathcal{T}_F(\rho_F(x, m)) \to \mathcal{T}_F(\mathcal{C}_F(l_0, me[f, x \mapsto m]))\}$
where $M_0 = \left\{ m \in M \mid Analyzes_m^F(ac) \right\}$.

The above definition clearly for each V determines a unique union type $\mathcal{T}_F(V)$, since recursion is "beneath a constructor" and since $FlowSet_F$ is finite (ensuring regularity).

EXAMPLE 6.1. With terminology as in Examples 3.6 and 4.1, it is easy to see—provided that $q_{\mathtt{x}}$ is considered another name for $ac_{\mathtt{x}}$—first that $\mathcal{T}_{\mathsf{F}_{2.1}}(\{v'_{\mathtt{x}}\}) = u'_{\mathtt{x}}$, and then that $\mathcal{T}_{\mathsf{F}_{2.1}}(\{v_{\mathtt{x}}\}) = u_{\mathtt{x}}$ since $\mathcal{T}_{\mathsf{F}_{2.1}}(\{v_{\mathtt{x}}\}).q_{\mathtt{x}}$ can be found as

$$\begin{aligned} & \bigwedge(\{1\} : \mathcal{T}_{\mathsf{F}_{2.1}}(\rho_{\mathsf{F}_{2.1}}(\mathtt{x},1)) \to \mathcal{T}_{\mathsf{F}_{2.1}}(\mathcal{C}_{\mathsf{F}_{2.1}}(7, me_{\mathtt{x}1})), \{2\} : \mathcal{T}_{\mathsf{F}_{2.1}}(\rho_{\mathsf{F}_{2.1}}(\mathtt{x},2)) \to \mathcal{T}_{\mathsf{F}_{2.1}}(\mathcal{C}_{\mathsf{F}_{2.1}}(7, me_{\mathtt{x}2}))) \\ = \; & \bigwedge(\{1\} : \mathcal{T}_{\mathsf{F}_{2.1}}(\{\mathrm{Int}\}) \to \mathcal{T}_{\mathsf{F}_{2.1}}(\{\mathrm{Int}\}), \{2\} : \mathcal{T}_{\mathsf{F}_{2.1}}(\{v'_{\mathtt{x}}\}) \to \mathcal{T}_{\mathsf{F}_{2.1}}(\{v'_{\mathtt{x}}\})) \\ = \; & \bigwedge(\{1\} : u_{\mathrm{int}} \to u_{\mathrm{int}}, \{2\} : u'_{\mathtt{x}} \to u'_{\mathtt{x}}). \end{aligned}$$

Note that without the M component in a flow value (ac, M), $v_{\mathtt{x}}$ would equal $v'_{\mathtt{x}}$ causing $\mathcal{T}_{\mathsf{F}_{2.1}}(\{v_{\mathtt{x}}\})$ to be an infinite type (as in P&P). □

Lemma 6.2. *The function $\mathcal{T}_F$ is monotone.* □

Proof. Assume that $V \leq_V V'$. Let $q \in dom(\mathcal{T}_F(V))$ be given; we must show that

$$q \in dom(\mathcal{T}_F(V')) \text{ with } \mathcal{T}_F(V).q \leq_t \mathcal{T}_F(V').q.$$

First assume that $q = q_{\mathrm{int}}$ so that $\mathcal{T}_F(V).q = \mathtt{int}$; then $\mathrm{Int} \in V$ so also $\mathrm{Int} \in V'$ implying $q \in dom(\mathcal{T}_F(V'))$ with $\mathcal{T}_F(V').q = \mathtt{int}$.

Next assume that $q \neq q_{\mathrm{int}}$; that is $q = ac$ for some ac with the property that there exists M such that $(ac, M) \in V$. Since $V \leq_V V'$ there exists M' with $M' \subseteq M$ such that $(ac, M') \in V'$, showing that $q \in dom(\mathcal{T}_F(V'))$. For each $m \in M$ there exists u_m and u'_m such that

$$\begin{aligned} &\mathcal{T}_F(V).q = \textstyle\bigwedge_{m \in M_0}\{\{m\} : u_m \to u'_m\} \text{ where } M_0 = \left\{m \in M \mid Analyzes^F_m(ac)\right\} \\ &\mathcal{T}_F(V').q = \textstyle\bigwedge_{m \in M'_0}\{\{m\} : u_m \to u'_m\} \text{ where } M'_0 = \left\{m \in M' \mid Analyzes^F_m(ac)\right\}. \end{aligned}$$

Since $M'_0 \subseteq M_0$, this demonstrates the desired relation $\mathcal{T}_F(V).q \leq_t \mathcal{T}_F(V').q$. □

Lemma 6.3. $\mathcal{T}_F(\{\mathit{Int}\}) = u_{\mathrm{int}}$. □

For z and m such that $(z, m) \in Reach^F_P$, we define $\mathcal{T}^\rho_F(z, m)$ as $\mathcal{T}_F(\rho_F(z, m))$ (by Lemma 4.16 this is well-defined). And for $e = ue^l$ and me such that $(e, me) \in Reach^F_P$, we construct a judgement $\mathcal{T}^J_F(e, me)$ as

$$\mathcal{T}^A_F(me) \vdash e : \mathcal{T}_F(\mathcal{C}_F(l, me))$$

where $\mathcal{T}^A_F(me)$ is defined recursively by $\mathcal{T}^A_F(\epsilon) = \epsilon$ and $\mathcal{T}^A_F(me[z \mapsto m]) = \mathcal{T}^A_F(me)[z \mapsto \mathcal{T}^\rho_F(z, m)]$ (by Lemma 4.16 also this is well-defined).

Definition 6.4. With F a flow analysis for P, the typing $T = \mathcal{T}(F)$ is given by $\langle P, Mem_F, AbsClos_F \cup \{q_{\mathrm{int}}\}, D_T\rangle$, where D_T is defined by stipulating that whenever (e, me) is in $Reach^F_P$ then D_T contains $\mathcal{T}^J_F(e, me)$, and that $\mathcal{T}^J_F(e', me')$ is a premise of $\mathcal{T}^J_F(e, me)$ iff $(e, me) \in Reach^F_P$ is among the immediate conditions (cf. the definition of $Reach^F_P$) for $(e', me') \in Reach^F_P$. □

EXAMPLE 6.5. It is easy to check that $\mathsf{T}_{2.1} = \mathcal{T}(\mathsf{F}_{2.1})$ and that $\mathsf{T}_{2.2} = \mathcal{T}(\mathsf{F}_{2.2})$, modulo renaming of the U-tags. □

Clearly D_T is a tree-formed derivation, and $\mathcal{T}^J_F(e, me)$ has address me in D_T. We must of course also prove that all judgements in D_T are in fact derivable from their premises using the inference rules in Fig. 6. This is the core of the following theorem, proved in Appendix D.

Theorem 6.6. *If F is valid and safe then $T = \mathcal{T}(F)$ as constructed by Definition 6.4 is a typing for P. The derivation D_T has the following properties:*

- *if D_T contains at address me a judgement for $\mu f.\lambda x.e$, it is derived using* $[\mathsf{fun}]^{w^\lambda}$ *where $w^\lambda = (ac : (\mathcal{T}_F(\{(ac, Mem_F)\})).ac)$ with $ac = (\mu f.\lambda x.e, me)$;*

- *if D_T contains at address me a judgement for $e_1 \,@\, ue_2^{l_2}$ with the leftmost premise of the form $A \vdash e_1 : u_1$, then it is derived using $[\mathsf{app}]^{w^@}$ where for all $q \in dom(u_1)$ it holds that $w^@(q) = \Phi_F((l_2, me), q)$.*

Finally, T is uniform with A_T given by $\mathcal{T}^\rho$ ☐

7 Round Trips

The two previous sections have provided translations $\mathcal{F}$ and $\mathcal{T}$ between derivations and flow analyses, and their correctness have been stated (Theorems 5.6 and 6.6). Next consider the "round-trip" translations $\mathcal{F} \circ \mathcal{T}$ (from flows to types and back) and $\mathcal{T} \circ \mathcal{F}$ (from types to flows and back). Both roundtrips are idempotent: they act as the identity on "canonical" elements, and otherwise "canonicalize".

Example 7.1. Exs. 5.5 and 6.5 illustrate that $\mathcal{F} \circ \mathcal{T}$ is the identity on $\mathsf{F}_{2.1}$ and $\mathsf{F}_{2.2}$, and that $\mathcal{T} \circ \mathcal{F}$ is the identity (modulo renaming of U-tags) on $\mathsf{T}_{2.1}$ and $\mathsf{T}_{2.2}$. In particular $\mathcal{T} \circ \mathcal{F}$ does not necessarily introduce infinite types, thus solving an open problem in P&P. ☐

7.1 Round Trips from the Flow World

The results below, to be proved in Appendix E, show that $\mathcal{F} \circ \mathcal{T}$ filters out everything that is not reachable, and acts as the identity ever after.

Theorem 7.2. *Assume that F is valid and safe for a program P, and let $F' = \mathcal{F}(\mathcal{T}(F))$. Then F' is valid and safe for P with $Mem_{F'} = Mem_F$, and*

- $Reach_P^{F'} = Reach_P^F$
- *$\mathcal{C}_{F'}(l, me) \neq \bot$ iff $\mathcal{C}_F(l, me) \neq \bot$ and $(ue^l, me) \in Reach_P^F$, in which case $\mathcal{C}_{F'}(l, me) = \mathrm{filter}_P^F(\mathcal{C}_F(l, me))$*
- *$\rho_{F'}(z, m) \neq \bot$ iff $\rho_F(z, m) \neq \bot$ and $(z, m) \in Reach_P^F$, in which case $\rho_{F'}(z, m) = \mathrm{filter}_P^F(\rho_F(z, m))$*
- *$\Phi_{F'}((l_2, me), ac) = K$ iff—with $ac = (\mu f.\lambda x.e_0, me_0)$ and with l_2 such that $e = ue_1^{l_1} \,@_l\, ue_2^{l_2}$ in $SubExpr_P$—it holds that $\Phi_F((l_2, me), ac) = K$ and $(e, me) \in Reach_P^F$ and $(\mu f.\lambda x.e_0, me_0) \in Reach_P^F$ and there exists M such that $(ac, M) \in \mathcal{C}_F(l_1, me)$.*

Here $\mathrm{filter}_P^F(V)$ is given by

$$\begin{aligned}&\{(ac, M') \mid (ac, M) \in V \text{ and } (\mu f.\lambda x.e_0, me_0) \in Reach_P^F\\ &\qquad \text{where } ac = (\mu f.\lambda x.e_0, me_0) \text{ and } M' = \left\{m \in M \mid (e_0, me_0[f, x \mapsto m]) \in Reach_P^F\right\}\\ &\cup (\text{if } Int \in V \text{ then } \{Int\} \text{ else } \emptyset)\end{aligned}$$

☐

Corollary 7.3. *Assume that F is valid and safe for a program P, let $F' = \mathcal{F}(\mathcal{T}(F))$, and let $F'' = \mathcal{F}(\mathcal{T}(F'))$. Then $F'' = F'$.* ☐

Clearly everything not reachable may be considered "junk". However, some junk is reachable and is hence not removed by $\mathcal{F} \circ \mathcal{T}$, as demonstrated by the following example. That our flow/type correspondence can faithfully encode such imprecisions shows the power of our framework.

Example 7.4. Consider the program P given by

$$(\lambda^1 \mathtt{x}.\mathtt{x}^2) \,@_0\, (\lambda^3 \mathtt{y}.\lambda^4 \mathtt{z}.\mathtt{z}^5)$$

$$\dfrac{\dfrac{[\mathtt{x} \mapsto u_{\mathtt{y}}] \vdash \mathtt{x}^2 : u_{\mathtt{y}}}{\epsilon \vdash \lambda^1\mathbf{x}.\mathtt{x}^2 : u_{\mathtt{x}}} \quad \epsilon \vdash \lambda^3\mathbf{y}.\lambda^4\mathbf{z}.\mathbf{z}^5 : u_{\mathtt{y}}}{\epsilon \vdash (\lambda^1\mathbf{x}.\mathtt{x}^2) @_0 (\lambda^3\mathbf{y}.\lambda^4\mathbf{z}.\mathbf{z}^5) : u_{\mathtt{y}}}$$

Figure 12: Example 7.4: the derivation D_T.

$$\dfrac{\dfrac{[\mathtt{x} \mapsto u_{\mathtt{y}}] \vdash \mathtt{x}^2 : u_{\mathtt{y}}}{\epsilon \vdash \lambda^1\mathbf{x}.\mathtt{x}^2 : u_{\mathtt{xz}}} \quad \epsilon \vdash \lambda^3\mathbf{y}.\lambda^4\mathbf{z}.\mathbf{z}^5 : u_{\mathtt{y}}}{\epsilon \vdash (\lambda^1\mathbf{x}.\mathtt{x}^2) @_0 (\lambda^3\mathbf{y}.\lambda^4\mathbf{z}.\mathbf{z}^5) : u_{\mathtt{y}}}$$

Figure 13: Example 7.4: the derivation $D_{T_{\mathtt{z}}}$.

and let

$$\begin{array}{lllllll} ac_{\mathtt{x}} & = & (\lambda\mathbf{x}.\mathtt{x}^2, \epsilon) & & v_{\mathtt{x}} & = & (ac_{\mathtt{x}}, \{\bullet\}) \\ ac_{\mathtt{y}} & = & (\lambda\mathbf{y}.\lambda^4\mathbf{z}.\mathbf{z}^5, \epsilon) & & v_{\mathtt{y}} & = & (ac_{\mathtt{y}}, \{\bullet\}) \\ ac_{\mathtt{z}} & = & (\lambda\mathbf{z}.\mathbf{z}^5, [\mathtt{y} \mapsto \bullet]) & & v_{\mathtt{z}} & = & (ac_{\mathtt{z}}, \{\bullet\}) \end{array}$$

By Theorem 4.9 there exists a least 0-CFA flow analysis F for P, and it is easy to see that F is given by the entries below:

$$\begin{array}{rcl} \{v_{\mathtt{y}}\} & = & \mathcal{C}_F(0,\epsilon) \;=\; \mathcal{C}_F(2,[\mathtt{x}\mapsto\bullet]) \;=\; \mathcal{C}_F(3,\epsilon) \;=\; \rho_F(\mathtt{x},\bullet) \\ \{v_{\mathtt{x}}\} & = & \mathcal{C}_F(1,\epsilon) \end{array}$$

The typing $T = \mathcal{T}(F)$ contains the derivation depicted in Fig. 12, where

$$\begin{array}{l} u_{\mathtt{y}} = \mathcal{T}_F(\{v_{\mathtt{y}}\}) = \bigvee(ac_{\mathtt{y}} : \bigwedge()) \\ u_{\mathtt{x}} = \mathcal{T}_F(\{v_{\mathtt{x}}\}) = \bigvee(ac_{\mathtt{x}} : \bigwedge(\{\bullet\} : u_{\mathtt{y}} \to u_{\mathtt{y}})) \end{array}$$

Note that T does not contain a judgement for $\lambda^4\mathbf{z}.\mathbf{z}^5$ since $(\lambda^4\mathbf{z}.\mathbf{z}^5, \epsilon)$ is not in Reach_P^F.

Next consider a 0-CFA flow analysis $F_{\mathtt{z}}$ where some junk that is *not* reachable has been added: $F_{\mathtt{z}}$ is as F except that

$$\begin{array}{l} \mathcal{C}_{F_{\mathtt{z}}}(1,\epsilon) = \{v_{\mathtt{x}}, v_{\mathtt{z}}\} \\ \rho_{F_{\mathtt{z}}}(\mathtt{z},\bullet) = \mathcal{C}_{F_{\mathtt{z}}}(5,[\mathtt{z}\mapsto\bullet]) = \{v_{\mathtt{y}}\}. \end{array}$$

The typing $T_{\mathtt{z}} = \mathcal{T}(F_{\mathtt{z}})$ contains the derivation depicted in Fig. 13, where

$$\begin{array}{l} u_{\mathtt{y}} = \mathcal{T}_{F_{\mathtt{z}}}(\{v_{\mathtt{y}}\}) = \bigvee(ac_{\mathtt{y}} : \bigwedge()) \\ u_{\mathtt{xz}} = \mathcal{T}_{F_{\mathtt{z}}}(\{v_{\mathtt{x}}, v_{\mathtt{z}}\}) = \bigvee(ac_{\mathtt{x}} : \bigwedge(\{\bullet\} : u_{\mathtt{y}} \to u_{\mathtt{y}}), ac_{\mathtt{z}} : \bigwedge(\{\bullet\} : u_{\mathtt{y}} \to u_{\mathtt{y}})) \end{array}$$

Now it is easy to see that $\mathcal{F}_{T_{\mathtt{z}}}(u_{\mathtt{y}}) = \{v_{\mathtt{y}}\}$ and $\mathcal{F}_{T_{\mathtt{z}}}(u_{\mathtt{xz}}) = \{v_{\mathtt{x}}\}$, implying that $\mathcal{F}(\mathcal{T}(F_{\mathtt{z}})) = F$. This illustrates that $\mathcal{F} \circ \mathcal{T}$ removes junk that is not reachable.

Finally consider a 0-CFA flow analysis $F_{\mathtt{y}}$ where some junk that *is* reachable has been added: $F_{\mathtt{y}}$ is given by the entries below:

$$\begin{array}{rcl} \{v_{\mathtt{y}}\} & = & \mathcal{C}_{F_{\mathtt{y}}}(2,[\mathtt{x}\mapsto\bullet]) \;=\; \mathcal{C}_{F_{\mathtt{y}}}(3,\epsilon) \;=\; \rho_{F_{\mathtt{y}}}(\mathtt{x},\bullet) \;=\; \rho_{F_{\mathtt{y}}}(\mathtt{y},\bullet) \\ \{v_{\mathtt{x}}, v_{\mathtt{y}}\} & = & \mathcal{C}_{F_{\mathtt{y}}}(1,\epsilon) \\ \{v_{\mathtt{z}}\} & = & \mathcal{C}_{F_{\mathtt{y}}}(4,[\mathtt{y}\mapsto\bullet]) \\ \{v_{\mathtt{y}}, v_{\mathtt{z}}\} & = & \mathcal{C}_{F_{\mathtt{y}}}(0,\epsilon) \end{array}$$

The typing $T_{\mathtt{y}} = \mathcal{T}(F_{\mathtt{y}})$ contains the derivation depicted in Fig. 14, where

$$\begin{array}{l} u_{\mathtt{z}} = \mathcal{T}_{F_{\mathtt{y}}}(\{v_{\mathtt{z}}\}) = \bigvee(ac_{\mathtt{z}} : \bigwedge()) \\ u_{\mathtt{y}} = \mathcal{T}_{F_{\mathtt{y}}}(\{v_{\mathtt{y}}\}) = \bigvee(ac_{\mathtt{y}} : \bigwedge(\{\bullet\} : u_{\mathtt{y}} \to u_{\mathtt{z}})) \\ u_{\mathtt{xy}} = \mathcal{T}_{F_{\mathtt{y}}}(\{v_{\mathtt{x}}, v_{\mathtt{y}}\}) = \bigvee(ac_{\mathtt{x}} : \bigwedge(\{\bullet\} : u_{\mathtt{y}} \to u_{\mathtt{y}}), ac_{\mathtt{y}} : \bigwedge(\{\bullet\} : u_{\mathtt{y}} \to u_{\mathtt{z}})) \\ u_{\mathtt{yz}} = \mathcal{T}_{F_{\mathtt{y}}}(\{v_{\mathtt{y}}, v_{\mathtt{z}}\}) = \bigvee(ac_{\mathtt{y}} : \bigwedge(\{\bullet\} : u_{\mathtt{y}} \to u_{\mathtt{z}}), ac_{\mathtt{z}} : \bigwedge()) \end{array}$$

$$\dfrac{\dfrac{[\mathtt{x} \mapsto u_{\mathtt{y}}] \vdash \mathtt{x}^2 \,:\, u_{\mathtt{y}}}{\epsilon \vdash \lambda^1\mathtt{x}.\mathtt{x}^2 \,:\, u_{\mathtt{xy}}} \quad \dfrac{[\mathtt{y} \mapsto u_{\mathtt{y}}] \vdash \lambda^4\mathtt{z}.\mathtt{z}^5 \,:\, u_{\mathtt{z}}}{\epsilon \vdash \lambda^3\mathtt{y}.\lambda^4\mathtt{z}.\mathtt{z}^5 \,:\, u_{\mathtt{y}}}}{\epsilon \vdash (\lambda^1\mathtt{x}.\mathtt{x}^2) \,@_0\, (\lambda^3\mathtt{y}.\lambda^4\mathtt{z}.\mathtt{z}^5) \,:\, u_{\mathtt{yz}}}$$

Figure 14: Example 7.4: the derivation $D_{T_{\mathtt{y}}}$.

Now it is easy to see that

$$\begin{aligned} &\mathcal{F}_{T_{\mathtt{y}}}(u_{\mathtt{z}}) = \{v_{\mathtt{z}}\} \\ &\mathcal{F}_{T_{\mathtt{y}}}(u_{\mathtt{y}}) = \{v_{\mathtt{y}}\} \\ &\mathcal{F}_{T_{\mathtt{y}}}(u_{\mathtt{xy}}) = \{v_{\mathtt{x}}, v_{\mathtt{y}}\} \\ &\mathcal{F}_{T_{\mathtt{y}}}(u_{\mathtt{yz}}) = \{v_{\mathtt{y}}, v_{\mathtt{z}}\} \end{aligned}$$

implying that $\mathcal{F}(\mathcal{T}(F_{\mathtt{y}})) = F_{\mathtt{y}}$. This illustrates that $\mathcal{F} \circ \mathcal{T}$ does not remove junk that is reachable. □

The above example also illustrates that $\mathcal{T} \circ \mathcal{F} \circ \mathcal{T} = \mathcal{T}$ does *not* in general hold, since

$$\mathcal{T}(\mathcal{F}(\mathcal{T}(F_{\mathtt{z}}))) \;=\; \mathcal{T}(F) \;=\; T \;\neq\; T_{\mathtt{z}} \;=\; \mathcal{T}(F_{\mathtt{z}}).$$

7.2 Round Trips from the Type World

The canonical typings are the ones that are *strongly consistent*:

Definition 7.5. A typing T is strongly consistent iff for all u that occur in[14] D_T and for all $q \in \mathit{dom}(u)$ with $q \neq q_{\mathrm{int}}$ the following holds:

> D_T contains exactly one judgement derived by an application of $[\mathsf{fun}]^{w^\lambda}$ with w^λ taking the form $(q : t)$, and this t satisfies $t \leq^c_\wedge u.q$.

Here $\leq^c_\wedge$ is a subrelation of $\leq_t$, defined by stipulating that $\mathtt{int} \leq^c_\wedge \mathtt{int}$ and that

> $\bigwedge_{i \in I}\{K_i : u_i \to u'_i\} \leq^c_\wedge \bigwedge_{i \in I_0}\{K_i : u_i \to u'_i\}$ iff $I_0 \subseteq I$.

□

Theorem 7.6. *Assume that T is a uniform typing for a program P, and let $T' = \mathcal{T}(\mathcal{F}(T))$. Then T' is a uniform typing for P with $IT_{T'} = IT_T$, and*

- *$D_{T'}$ contains a judgement for e with address ke iff D_T contains a judgement for e with address ke (i.e., the two derivations have the same shape);*
- *$D_{T'}$ is strongly consistent;*
- *if D_T is strongly consistent then $D_{T'}$ equals D_T (modulo renaming of U-tags).*

□

Proof. See Appendix E. □

Example 7.7. We now again consider the motivating example put forward in Sect. 1, where the program

> $\big(\lambda^1\mathtt{x}.\mathtt{succ}\ \mathtt{x}\big) \,@\, \big((\lambda^2\mathtt{y}.\mathtt{y}) \,@\, 3\big)$

was given essentially the typing $\langle P, \{\bullet\}, \{\bullet\}, D_T\rangle$ with D_T as depicted in Fig. 15; we use $i \to i$ as a shorthand for $\bigwedge(\{\bullet\} : u_{\mathrm{int}} \to u_{\mathrm{int}})$. Note that T is *not* strongly consistent.

The flow analysis $F = \mathcal{F}(T)$ is given by

[14] That is, the u with the property that there exists u' with u a subtree of u' such that D_T contains a judgement $A_0 \vdash e_0 : u_0$ with $u' = u_0$ or $(z \mapsto u') \in A_0$ for some z.

$$\dfrac{\dfrac{\dfrac{[\mathtt{x} \mapsto u_{\text{int}}] \vdash \mathtt{x} : u_{\text{int}}}{[\mathtt{x} \mapsto u_{\text{int}}] \vdash \mathtt{succ\ x} : u_{\text{int}}}}{\epsilon \vdash \lambda\mathtt{x}.\mathtt{succ\ x} : \bigvee(\bullet : i \to i)}\,[\mathsf{fun}]^{(\bullet:i\to i)} \qquad \dfrac{\dfrac{[\mathtt{y} \mapsto u_{\text{int}}] \vdash \mathtt{y} : u_{\text{int}}}{\epsilon \vdash \lambda\mathtt{y}.\mathtt{y} : \bigvee(\bullet : i \to i)}\,[\mathsf{fun}]^{(\bullet:i\to i)} \qquad \epsilon \vdash 3 : u_{\text{int}}}{\epsilon \vdash \left(\lambda^2\mathtt{y}.\mathtt{y}\right) @ 3 : u_{\text{int}}}}{\epsilon \vdash \left(\lambda^1\mathtt{x}.\mathtt{succ\ x}\right) @ \left(\left(\lambda^2\mathtt{y}.\mathtt{y}\right) @ 3\right) : u_{\text{int}}}$$

Figure 15: Example 7.7: the derivation D_T.

$$\dfrac{\dfrac{\dfrac{[\mathtt{x} \mapsto u_{\text{int}}] \vdash \mathtt{x} : u_{\text{int}}}{[\mathtt{x} \mapsto u_{\text{int}}] \vdash \mathtt{succ\ x} : u_{\text{int}}}}{\epsilon \vdash \lambda\mathtt{x}.\mathtt{succ\ x} : u}\,[\mathsf{fun}]^{(q_1:i\to i)} \qquad \dfrac{\dfrac{[\mathtt{y} \mapsto u_{\text{int}}] \vdash \mathtt{y} : u_{\text{int}}}{\epsilon \vdash \lambda\mathtt{y}.\mathtt{y} : u}\,[\mathsf{fun}]^{(q_2:i\to i)} \qquad \epsilon \vdash 3 : u_{\text{int}}}{\epsilon \vdash (\lambda\mathtt{y}.\mathtt{y}) @ 3 : u_{\text{int}}}}{\epsilon \vdash (\lambda\mathtt{x}.\mathtt{succ\ x}) @ ((\lambda\mathtt{y}.\mathtt{y}) @ 3) : u_{\text{int}}}$$

Figure 16: Example 7.7: the derivation $D_{T'}$.

$$\mathcal{C}_F(1,\epsilon) = \mathcal{C}_F(2,\epsilon) = \mathcal{F}_T(\bigvee(\bullet : i \to i)) = \{((\lambda\mathtt{x}.\mathtt{succ\ x}, \epsilon), \{\bullet\}), ((\lambda\mathtt{y}.\mathtt{y}, \epsilon), \{\bullet\})\}$$

in that all other reachable entries in $\mathcal{C}_F$ and ρ_F are {Int}.

Then the typing $T' = \mathcal{T}(F)$ is given by $\langle P, \{\bullet\}, \{q_1, q_2\}, D_{T'}\rangle$, with $q_1 = (\lambda\mathtt{x}.\mathtt{succ\ x}, \epsilon)$ and $q_2 = (\lambda\mathtt{y}.\mathtt{y}, \epsilon)$, and with $D_{T'}$ as depicted in Fig. 16 where u denotes $\bigvee(q_1 : i \to i, q_2 : i \to i)$. Note that T' is strongly consistent, as guaranteed by Theorem 7.6.

□

Again, the ability to faithfully encode both precise and imprecise analyses in both the type world and the flow world demonstrates the power of our framework.

8 Discussion

Our flow system follows the lines of N&N, generalizing some features while omitting others (such as polymorphic splitting [WJ98], left for future work). That it has substantial descriptive power is indicated by the fact that it encompasses both argument-based and call-string based polyvariance. In particular, the flow analysis framework of P&P, designed so as to model several existing flow analyses and on the surface quite different from ours, can be encoded into our framework. Unlike P&P, our flow logic has a subject reduction property, inherited from the N&N approach.

The generality of our type system is less clear. The annotation with tags gives rise to intersection and union types that are not associative, commutative, or idempotent (ACI). This stands in contrast to the ACI types of P&P, but is similar to the non-ACI intersection and union types of CIL, the intermediate language of an experimental compiler that integrates flow information into the type system [WDMT97a, DMTW97]. Indeed, a key motivation of this work was to formalize the encoding of various flow analyses in the CIL type system. Developing a translation between the the type system of this paper and CIL is our next goal.

Acknowledgments

Our work adds a new layer on top of foundations constructed by Jens Palsberg, Flemming Nielson, and Hanne Riis Nielson in their seminal work. We are fortunate to have had many interesting discussions on type systems and flow analysis with these authors, as well as with our Church Project compatriots, especially Assaf Kfoury, Joe Wells, Bob Muller, Allyn Dimock, and Anindya Banerjee. This paper grew out of an ESOP conference paper that benefited from the feedback of the referees.

References

[AC93] R. Amadio and L. Cardelli. Subtyping recursive types. *ACM Trans. on Prog. Langs. & Systs.*, 15(4):575–631, 1993.

[Age95] O. Agesen. The Cartesian product algorithm. In *Proceedings of ECOOP'95, Seventh European Conference on Object-Oriented Programming*, vol. 952, pp. 2–26. Springer-Verlag, 1995.

[AT00] T. Amtoft and F. Turbak. Faithful translations between polyvariant flows and polymorphic types. In ESOP '00 [ESOP00], pp. 26–40.

[Ban97] A. Banerjee. A modular, polyvariant, and type-based closure analysis. In ICFP '97 [ICFP97].

[CJW00] H. Cejtin, S. Jagannathan, and S. Weeks. Flow-directed closure conversion for typed languages. In ESOP '00 [ESOP00], pp. 56–71.

[DCG95] J. Dean, C. Chambers, and D. Grove. Selective specialization for object-oriented languages. In *Proc. ACM SIGPLAN '95 Conf. Prog. Lang. Design & Impl.*, 1995.

[DMTW97] A. Dimock, R. Muller, F. Turbak, and J. B. Wells. Strongly typed flow-directed representation transformations. In ICFP '97 [ICFP97], pp. 11–24.

[DWM+01] A. Dimock, I. Westmacott, R. Muller, F. Turbak, and J. B. Wells. Functioning without closure: Type-safe customized function representations for Standard ML. In *Proc. 2001 Int'l Conf. Functional Programming*. ACM Press, 2001.

[ESOP00] *Programming Languages & Systems, 9th European Symp. Programming*, vol. 1782 of *LNCS*. Springer-Verlag, 2000.

[Fax93] K.-F. Faxén. Cheap eagerness: Speculative evaluation in a lazy language. In *Proc. ACM SIGPLAN '93 Conf. Prog. Lang. Design & Impl.*, pp. 150–161, 1993.

[GNN97] K. L. S. Gasser, F. Nielson, and H. R. Nielson. Systematic realisation of control flow analyses for CML. In ICFP '97 [ICFP97], pp. 38–51.

[Hei95] N. Heintze. Control-flow analysis and type systems. In SAS '95 [SAS95], pp. 189–206.

[HH98] J. Hannan and P. Hicks. Higher-order uncurrying. In *Conf. Rec. POPL '98: 25th ACM Symp. Princ. of Prog. Langs.*, pp. 1–11, 1998.

[ICFP97] *Proc. 1997 Int'l Conf. Functional Programming*. ACM Press, 1997.

[JW95] S. Jagannathan and S. Weeks. A unified treatment of flow analysis in higher-order languages. In *Conf. Rec. 22nd Ann. ACM Symp. Princ. of Prog. Langs.*, pp. 393–407. ACM, 1995.

[JWW97] S. Jagannathan, S. Weeks, and A. Wright. Type-directed flow analysis for typed intermediate languages. In *Proc. 4th Int'l Static Analysis Symp.*, vol. 1302 of *LNCS*. Springer-Verlag, 1997.

[KPS95] D. Kozen, J. Palsberg, and M. I. Schwartzbach. Efficient recursive subtyping. *Mathematical Structures in Computer Science*, 5(1):113–125, 1995. Preliminary version in Proceedings of POPL'93, pages 419–428.

[MWCG99] G. Morrisett, D. Walker, K. Crary, and N. Glew. From System F to typed assembly language. *ACM Trans. on Prog. Langs. & Systs.*, 21(3):528–569, May 1999.

[NN97] F. Nielson and H. R. Nielson. Infinitary control flow analysis: A collecting semantics for closure analysis. In *Conf. Rec. POPL '97: 24th ACM Symp. Princ. of Prog. Langs.*, pp. 332–345, 1997.

[NN98] F. Nielson and H. R. Nielson. Flow logic and operational semantics. *Electronic Notes in Theoretical Computer Science*, 10, 1998.

[NN99] F. Nielson and H. R. Nielson. Interprocedural control flow analysis. In *Proc. European Symp. on Programming*, vol. 1576 of *LNCS*, pp. 20–39. Springer-Verlag, 1999.

[PC95] J. Plevyak and A. Chien. Type directed cloning for object-oriented programs. In *Workshop for Languages and Compilers for Parallel Computers*, Aug. 1995.

[PJ96] S. L. Peyton Jones. Compiling Haskell by program transformation: A report from the trenches. In *Proc. European Symp. on Programming*, 1996.

[PO95] J. Palsberg and P. O'Keefe. A type system equivalent to flow analysis. *ACM Trans. on Prog. Langs. & Systs.*, 17(4):576–599, 1995.

[PP01] J. Palsberg and C. Pavlopoulou. From polyvariant flow information to intersection and union types. *J. Funct. Programming*, 11(3):263–317, May 2001.

[SAS95] *Proc. 2nd Int'l Static Analysis Symp.*, vol. 983 of *LNCS*, 1995.

[Sch95] D. Schmidt. Natural-semantics-based abstract interpretation. In SAS '95 [SAS95], pp. 1–18.

[SGL96] V. C. Sreedhar, G. R. Gao, and Y. Lee. Identifying loops using DJ graphs. *ACM Trans. on Prog. Langs. & Systs.*, 18(6):649–658, 1996.

[Shi91] O. Shivers. *Control Flow Analysis of Higher Order Languages*. Ph.D. thesis, Carnegie Mellon University, 1991.

[SW97] P. Steckler and M. Wand. Lightweight closure conversion. *ACM Trans. on Prog. Langs. & Systs.*, 19(1):48–86, Jan. 1997.

[TAPS97] *Proc. 7th Int'l Joint Conf. Theory & Practice of Software Development*, 1997.

[TMC+96] D. Tarditi, G. Morrisett, P. Cheng, C. Stone, R. Harper, and P. Lee. TIL: A type-directed optimizing compiler for ML. In *Proc. ACM SIGPLAN '96 Conf. Prog. Lang. Design & Impl.*, 1996.

[TO98] A. P. Tolmach and D. Oliva. From ML to Ada: Strongly-typed language interoperability via source translation. *J. Funct. Programming*, 8(4):367–412, 1998.

[Tol97] A. Tolmach. Combining closure conversion with closure analysis using algebraic types. In *Proc. First Int'l Workshop on Types in Compilation*, June 1997.

[WDMT97a] J. B. Wells, A. Dimock, R. Muller, and F. Turbak. A typed intermediate language for flow-directed compilation. In TAPS '97 [TAPS97], pp. 757–771.

[WDMT97b] J. B. Wells, A. Dimock, R. Muller, and F. Turbak. A typed intermediate language for flow-directed compilation. In TAPS '97 [TAPS97], pp. 757–771. Superseded by [WDMT0X].

[WDMT0X] J. B. Wells, A. Dimock, R. Muller, and F. Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Programming*, 200X. To appear. Supersedes [WDMT97b].

[WJ98] A. Wright and S. Jagannathan. Polymorphic splitting: An effective polyvariant flow analysis. *ACM Trans. on Prog. Langs. & Systs.*, 20:166–207, 1998.

[WS99] M. Wand and I. Siveroni. Constraint systems for useless variable elimination. In *Conf. Rec. POPL '99: 26th ACM Symp. Princ. of Prog. Langs.*, pp. 291–302, 1999.

A Types

Proof of Lemma 3.4. First note that we can decompose the functional $\mathcal{H}$ into $\mathcal{H}_u$ and $\mathcal{H}_t$, that is $\mathcal{H}(Q_u,\ Q_t\ )$ equals $(\mathcal{H}_u(Q_t), \mathcal{H}_t(Q_u, Q_t))$; here

$$\bigvee_{i\in I}\{q_i : t_i\}\ \mathcal{H}_u(Q_t)\ \bigvee_{j\in J}\{q'_j : t'_j\} \text{ iff } \forall i \in I.\ \exists j\ \in\ J.\ q_i = q'_j \text{ and } t_i\ Q_t\ t'_j$$

and similarly for $\mathcal{H}_t(Q_u, .)$

Below, we shall want to prove

$$Q_u \subseteq \leq_u \text{ and } Q_t \subseteq \leq_t \tag{1}$$

for suitable choices of Q_u and Q_t. By the principle of coinduction, this can be done by establishing $(Q_u, Q_t) \subseteq \mathcal{H}(Q_u,\ Q_t\ )$ which amounts to showing

$$Q_u \subseteq \mathcal{H}_u(Q_t) \text{ and } Q_t \subseteq \mathcal{H}_t(Q_u, Q_t). \tag{2}$$

Reflexivity amounts to (1) with Q_u and Q_t defined as follows: $u\ Q_u\ u'$ holds iff $u = u'$ and $t\ Q_t\ t'$ holds iff $t = t'$. For this choice of Q_u and Q_t we must show (2). So first assume that $u\ Q_u\ u$, with $u = \bigvee_{i\in I}\{q_i : t_i\}$. Then for all $i \in I$ we have $q_i = q_i$ and $t_i\ Q_t\ t_i$, showing that $u\ \mathcal{H}_u(Q_t)\ u$. This establishes the first half of (2).

Next assume that $t\ Q_t\ t$. If $t = \texttt{int}$, we clearly have $t\ \mathcal{H}_t(Q_u, Q_t)\ t$. So assume that $t = \bigwedge_{i\in i}\{K_i : u_i \to u'_i\}$. Then for all $i \in I$ it holds with $I_0 = \{i\}$ that

$$K_i = \cup_{i_0\in I_0} K_{i_0} \text{ and } \forall i_0 \in I_0.\ u'_{i_0}\ Q_u\ u'_i \text{ and}$$
$$\forall q \in \mathit{dom}(u_i).\ \exists i_0\ \in\ I_0.\ q \in \mathit{dom}(u_{i_0}) \text{ and } u_i.q\ Q_t\ u_{i_0}.q.$$

So also here we have $t\ \mathcal{H}_t(Q_u, Q_t)\ t$. This establishes the second half of (2).

Transitivity amounts to (1) with Q_u and Q_t defined as follows: $u\ Q_u\ u'$ holds iff there exists u'' with $u \leq_u u''$ and $u'' \leq_u u'$; $t\ Q_t\ t'$ holds iff there exists t'' with $t \leq_t t''$ and $t'' \leq_t t'$. For this choice of Q_u and Q_t we must show (2). So first assume that $u \leq_u u'' \leq_u u'$, with $u = \bigvee_{i\in I}\{q_i : t_i\}$ and with $u' = \bigvee_{i\in I'}\{q'_i : t'_i\}$ and with $u'' = \bigvee_{i\in I''}\{q''_i : t''_i\}$. Let $i \in I$ be given. Since $u \leq_u u''$ there exists $i'' \in I''$ such that $q_i = q''_{i''}$ and $t_i \leq_t t''_{i''}$, and since $u'' \leq_u u'$ there then exists $i' \in I'$ such that $q''_{i''} = q'_{i'}$ and $t''_{i''} \leq_t t'_{i'}$. This shows that $q_i = q'_{i'}$ and $t_i\ Q_t\ t'_{i'}$, which amounts to the first half of (2).

Next assume that $t \leq_t t'' \leq_t t'$. If $t = \texttt{int}$ then clearly $t'' = \texttt{int}$ and $t' = \texttt{int}$, establishing the second half of (2). So assume that t takes the form $\bigwedge_{i\in I}\{K_i : u_{0i} \to u_{1i}\}$; then t'' takes the form $\bigwedge_{i\in I''}\{K''_i : u''_{0i} \to u''_{1i}\}$ and t' takes the form $\bigwedge_{i\in I'}\{K'_i : u'_{0i} \to u'_{1i}\}$. Let $i' \in I'$ be given. Since $t'' \leq_t t'$ there exists $J \subseteq I''$ such that

$$K'_{i'} = \cup_{j\in J} K''_j \text{ and } \forall j \in J.\ u''_{1j} \leq_u u'_{1i'} \text{ and}$$
$$\forall q \in \mathit{dom}(u'_{0i'}).\ \exists j\ \in\ J.\ q \in \mathit{dom}(u''_{0j}) \text{ and } u'_{0i'}.q \leq_t u''_{0j}.q.$$

Since $t \leq_t t''$, for all $j \in J$ there exists $I_j \subseteq I$ such that

$$K''_j = \cup_{i\in I_j} K_i \text{ and } \forall i \in I_j.\ u_{1i} \leq_u u''_{1j} \text{ and}$$
$$\forall q_j \in \mathit{dom}(u''_{0j}).\ \exists i\ \in\ I_J.\ q_j \in \mathit{dom}(u_{0i}) \text{ and } u''_{0j}.q_j \leq_t u_{0i}.q_j.$$

Let $I_0 = \cup_{j \in J} I_j$. Then $I_0 \subseteq I$, and

$$K'_{i'} = \cup_{i \in I_0} K_i \text{ and } \forall i \in I_0.\, u_{1i}\; Q_u\; u'_{1i'} \text{ and} \qquad (3)$$
$$\forall q \in \mathit{dom}(u'_{0i'}).\ \exists j \in J.\ \exists i \in I_J.\ q \in \mathit{dom}(u_{0i}) \text{ and } u'_{0i'}.q \leq_t u''_{0j}.q \leq_t u_{0i}.q$$

where the latter line implies that

$$\forall q \in \mathit{dom}(u'_{0i'}).\ \exists i \in I_0.\ q \in \mathit{dom}(u_{0i}) \text{ and } u'_{0i'}.q\; Q_t\; u_{0i}.q. \qquad (4)$$

Now (3) and (4) demonstrates the second half of (2). □

Proof of Lemma 3.8. As a preparation, we state a number of rather trivial facts (some proved by induction in the derivation):

Lemma A.1. *If $A \vdash e : u$ and $u \leq_u u'$ then also $A \vdash e : u'$.* □

Lemma A.2. *Assume that for all $z \in FV_{\mathrm{e}}(e)$ it holds that $A(z) = A'(z)$. Then $A \vdash e : u$ implies $A' \vdash e : u$.* □

Lemma A.3. *Suppose that $A \vdash \mu f.\lambda x.e : u$ is derived using $[\mathsf{fun}]^{(q:t)}$. Then also $A \vdash \mu f.\lambda x.e : \bigvee(q : t)$.* □

Lemma A.4. *If $A \vdash sv : u$ then there exists $q \in \mathit{dom}(u)$ such that $A \vdash sv : \bigvee(q : u.q)$.* □

We are now ready to prove Theorem 3.8, which facilitates a proof by induction in the derivation of $se \vdash e \Rightarrow e'$. We perform a case analysis in the rule applied; first we consider the structural cases:

(app_l). The situation is that $se \vdash e_1 \mathbin{@_l} e_2 \Rightarrow e'_1 \mathbin{@_l} e_2$ because $se \vdash e_1 \Rightarrow e'_1$. The premises of the judgement $A \vdash e_1^l e_2 : u$ are of the form

$$A \vdash e_1 : u_1 \text{ and } A \vdash e_2 : u_2.$$

The left premise shows that we can apply the induction hypothesis on the inference $se \vdash e_1 \Rightarrow e'_1$, yielding $A \vdash e'_1 : u_1$. Together with the right premise this shows that $A \vdash {e'_1}^l e_2 : u$. We have exploited that the condition for applying $[\mathsf{app}]^{w^@}$ depends solely on the union types (and not on the expressions).

The cases (app_r), ($succ$), and (if) are similar. The remaining cases are treated below:

(var). The situation is that $se \vdash z^l \Rightarrow sv^l$ with $sv = se(z)$. Our assumptions are that $se \bowtie A$, implying $\epsilon \vdash sv : A(z)$, and that $A \vdash z : u$, implying $A(z) \leq_u u$. By Lemmas A.1 and A.2 this demonstrates the desired judgement $A \vdash sv : u$.

(fun). The situation is that $se \vdash \mu f.\lambda^l x.e \Rightarrow \mathtt{close}^l\ \mathit{fn}\ \mathtt{in}\ se$, where $\mathit{fn} = \mu f.\lambda x.e$. Our assumption is that $se \bowtie A$ and that $A \vdash \mu f.\lambda^l x.e : u$. But this shows the desired judgement $A \vdash \mathtt{close}^l\ \mathit{fn}\ \mathtt{in}\ se : u$.

($succ_v$). The situation is that $se \vdash \mathtt{succ}^l\, c^{l_0} \Rightarrow c_1^l$. Our assumption is that $A \vdash \mathtt{succ}^l\, c^{l_0} : u$, implying $u_{\mathrm{int}} \leq_u u$. Then clearly also $A \vdash c_1 : u$, as desired.

(if_0). (the case ($if_>$) is similar.) The situation is, with $e = \mathtt{if0}^l\ 0^{l_0}\ \mathtt{then}\ ue_1^{l_1}\ \mathtt{else}\ e_2$, that $se \vdash e \Rightarrow ue_1^l$. Among the premises of the judgement $A \vdash e : u$ is a judgement of the form $A \vdash ue_1^{l_1} : u_1$, where $u_1 \leq_u u$. By Lemma A.1, this shows that $A \vdash ue_1^l : u$ as desired.

($bind$). The situation is that $se \vdash \mathtt{bind}^l\ se_1\ \mathtt{in}\ e_1 \Rightarrow \mathtt{bind}^l\ se_1\ \mathtt{in}\ e'_1$ because $se_1 \vdash e_1 \Rightarrow e'_1$. Our assumptions are that $A \vdash \mathtt{bind}^l\ se_1\ \mathtt{in}\ e_1 : u$, implying that there exists A_1 with $se_1 \bowtie A_1$ and $u_1 \leq_u u$ such that $A_1 \vdash e_1 : u_1$. We can thus apply the induction hypothesis on the inference $se_1 \vdash e_1 \Rightarrow e'_1$, yielding $A_1 \vdash e'_1 : u_1$. This demonstrates $A \vdash \mathtt{bind}^l\ se_1\ \mathtt{in}\ e'_1 : u$, as desired.

$\underline{(bind_v)}$. The situation is that $se \vdash \texttt{bind}^l\ se_1\ \texttt{in}\ sv^{l_1} \Rightarrow sv^l$. Our assumptions are that $A \vdash \texttt{bind}^l\ se_1\ \texttt{in}\ sv^{l_1} : u$, implying that there exists A_1 and $u_1 \leq_u u$ such that $A_1 \vdash sv : u_1$. By Lemmas A.1 and A.2 (recall that $FV_{\mathrm{e}}(sv) = \emptyset$) we infer the desired judgement $A \vdash sv : u$.

$\underline{(app_v)}$. The situation is, with $sv_1 = \texttt{close}\ fn_1\ \texttt{in}\ se_1$ and $fn_1 = \mu f.\lambda x.e_1$ that

$$se \vdash sv_1^{l_1} @_l sv_2^{l_2} \Rightarrow \texttt{bind}^l\ se_1[f \mapsto sv_1][x \mapsto sv_2]\ \texttt{in}\ e_1.$$

Our assumption is that $A \vdash sv_1 @_l sv_2 : u$. Let $w^@$, u_1 and u_2 be such that this judgement is derived by $[\mathsf{app}]^{w^@}$ from

$$A \vdash sv_1 : u_1 \tag{5}$$
$$A \vdash sv_2 : u_2 \tag{6}$$

employing that

$$\forall q_1 \in dom(u_1).\, u_1.q_1 \leq_t \bigwedge(w^@(q_1) : u_2 \to u). \tag{7}$$

From (5) we see that there exists A_1 with

$$se_1 \bowtie A_1 \tag{8}$$

such that $A_1 \vdash fn_1 : u_1$; there exists q and

$$t = \bigwedge_{k \in K}\{\{k\} : u_k \to u'_k\}$$

such that this judgement is derived using $[\mathsf{fun}]^{(q:t)}$. That is,

$$\bigvee(q : t) \leq_u u_1 \tag{9}$$

and for all $k \in K$ there exists u''_k with

$$\bigvee(q : t) \leq_u u''_k \tag{10}$$

such that one can derive

$$A_1[f \mapsto u''_k][x \mapsto u_k] \vdash e_1 : u'_k. \tag{11}$$

By Lemma A.3 it holds that $A_1 \vdash fn_1 : \bigvee(q : t)$, implying $A \vdash sv_1 : \bigvee(q : t)$; so from (10) we by Lemmas A.1 and A.2 infer

$$\forall k \in K.\, \epsilon \vdash sv_1 : u''_k. \tag{12}$$

From (6) we by Lemma A.4 infer that there exists $q_2 \in dom(u_2)$ such that

$$A \vdash sv_2 : \bigvee(q_2 : u_2.q_2). \tag{13}$$

(7) and (9) implies $t \leq_t \bigwedge(w^@(q) : u_2 \to u)$ from which we deduce that there exists $k_0 \in K$ such that

$$u'_{k_0} \leq_u u \tag{14}$$

and such that $u_2.q_2 \leq_t u_{k_0}.q_2$ which together with (13) demonstrates (using Lemmas A.1 and A.2) that

$$\epsilon \vdash sv_2 : u_{k_0}. \tag{15}$$

(8), (12) and (15) demonstrates that

$$se_1[f \mapsto sv_1][x \mapsto sv_2] \bowtie A_1[f \mapsto u''_{k_0}][x \mapsto u_{k_0}].$$

Together with (11) and (14) this demonstrates the desired judgement

$$A \vdash \texttt{bind}^l\ se_1[f \mapsto sv_1][x \mapsto sv_2]\ \texttt{in}\ e_1\ :\ u.$$

This completes the proof. □

Proof of Lem. 3.11. Our task is to prove that each inference in D_T is justified. Three cases:

$[\mathsf{var}]^{AC}$ has been applied. The inference takes the form

$$\mathcal{T}^{AC}(B) \vdash z^l\ :\ \mathcal{T}^{AC}(s)$$

where $B(z) \leq^{AC} s$. The monotonicity of $\mathcal{T}^{AC}$ thus yields the desired

$$\mathcal{T}^{AC}(B)(z) = \mathcal{T}^{AC}(B(z)) \leq_u \mathcal{T}^{AC}(s).$$

$[\mathsf{fun}]^{AC}$ has been applied. The inference takes the form

$$\frac{\mathcal{T}^{AC}(B)[f \mapsto \mathcal{T}^{AC}(s_0)][x \mapsto \mathcal{T}^{AC}(s_1)] \vdash e\ :\ \mathcal{T}^{AC}(s_2)}{\mathcal{T}^{AC}(B) \vdash \mu f.\lambda^l x.e\ :\ \mathcal{T}^{AC}(s)}$$

where $s_1 \to s_2 \leq^{AC} s_0$ and $s_1 \to s_2 \leq^{AC} s$.

Let $t = \mathcal{T}^{AC}(s_1) \to \mathcal{T}^{AC}(s_2)$ and $q = s_1 \to s_2$. Since $q \in UT_T$ with $q \leq^{AC} s_0$ and $q \leq^{AC} s$, we infer that the union types $\mathcal{T}^{AC}(s)$ and $\mathcal{T}^{AC}(s_0)$ both contain a component $(q : \mathcal{T}_0^{AC}(q))$, that is $(q : t)$. This shows the desired relations

$$\bigvee(q : t) \leq_u \mathcal{T}^{AC}(s) \text{ and } \bigvee(q : t) \leq_u \mathcal{T}^{AC}(s_0).$$

$[\mathsf{app}]^{AC}$ has been applied. The inference takes the form

$$\frac{\mathcal{T}^{AC}(B) \vdash e_1\ :\ \mathcal{T}^{AC}(s_1) \quad \mathcal{T}^{AC}(B) \vdash e_2\ :\ \mathcal{T}^{AC}(s_2)}{\mathcal{T}^{AC}(B) \vdash e_1 @_l e_2\ :\ \mathcal{T}^{AC}(s)}$$

where $s_1 \leq^{AC} s_2 \to s$.

Let $q \in dom(\mathcal{T}^{AC}(s_1))$, that is $q \leq^{AC} s_1$ and therefore $q \leq^{AC} s_2 \to s$. We then have the desired relation (where trivially, $w^{@}(q) = \{\bullet\}$)

$$\mathcal{T}^{AC}(s_1).q = \mathcal{T}_0^{AC}(q) \leq_t \mathcal{T}^{AC}(s_2) \to \mathcal{T}^{AC}(s).$$

□

B Flows

B.1 Subject reduction

Proof of Lemma 4.4. As a preparation, we state a number of rather trivial[15] facts:

Lemma B.1. *If $sv\ \mathcal{V}_F\ V$ and $V \leq_V V'$ then also $sv\ \mathcal{V}_F\ V'$.* □

Lemma B.2. *If $sv\ \mathcal{V}_F\ V$ then there exists $v \in V$ such that $sv\ \mathcal{V}_F\ \{v\}$.* □

Lemma B.3. *$sv\ \mathcal{V}_F\ \mathcal{C}_F(l, me)$ iff $F \models^{me} sv^l$.* □

Lemma B.4. *If $F \models^{me} ue^l$ and $\mathcal{C}_F(l, me) \leq_V \mathcal{C}_F(l', me)$ then $F \models^{me} ue^{l'}$.* □

We are now ready to prove Theorem 4.4, which facilitates a proof by induction in the derivation of $se \vdash e \Rightarrow e'$. We perform a case analysis in the rule applied; first we consider the structural cases:

(app_l). The situation is that $se \vdash e_1 @_l e_2 \Rightarrow e_1' @_l e_2$ because $se \vdash e_1 \Rightarrow e_1'$, where there exists l_1, ue_1 and ue_1' such that $e_1 = ue_1^{l_1}$ and $e_1' = {ue_1'}^{l_1}$. By assumption we have $se\ \mathcal{R}_F\ me$ and that

[15] For Lemma B.4 we exploit that in [*app*], Φ_F is given l_2 rather than l as argument.

$$F \models^{me} e_1 \mathbin{@_l} e_2. \tag{1}$$

(1) implies $F \models^{me} e_1$ so by applying the induction hypothesis on the inference $se \vdash e_1 \Rightarrow e_1'$ we infer $F \models^{me} e_1'$, which together with (1) demonstrates the desired relation $F \models^{me} e_1' \mathbin{@_l} e_2$. We have exploited that in the clause [*app*], the only condition that depends on ue_1 is "$F \models^{me} ue_1^{l_1}$".

The cases (app_r), (*succ*), and (*if*) are similar. The remaining cases are treated below:

<u>(*var*).</u> The situation is that $se \vdash z^l \Rightarrow sv^l$ with $sv = se(z)$. Since $se\ \mathcal{R}_F\ me$ we have $sv\ \mathcal{V}_F\ \rho_F(z, me(z))$; and since $F \models^{me} z^l$ we have $\rho_F(z, me(z)) \leq_V \mathcal{C}_F(l, me)$. By Lemma B.1 this shows $sv\ \mathcal{V}_F\ \mathcal{C}_F(l, me)$, which by Lemma B.3 amounts to the desired $F \models^{me} sv^l$.

<u>(*fun*).</u> The situation is that $se \vdash \mu f.\lambda^l x.e \Rightarrow \mathtt{close}^l\ fn\ \mathtt{in}\ se$, where $fn = \mu f.\lambda x.e$. Our assumption is that $se\ \mathcal{R}_F\ me$, and that $F \models^{me} \mu f.\lambda^l x.e$ which amounts to $\{((fn, me), Mem_F)\} \leq_V \mathcal{C}_F(l, me)$. This demonstrates that $F \models^{me} \mathtt{close}^l\ fn\ \mathtt{in}\ se$, as desired.

<u>($succ_v$).</u> The situation is that $se \vdash \mathtt{succ}^l\ c^{l_0} \Rightarrow c_1^l$. By assumption we have $F \models^{me} \mathtt{succ}^l\ c^{l_0}$ implying $\mathrm{Int} \in \mathcal{C}_F(l, me)$, showing $F \models^{me} c_1^l$ as desired.

<u>(if_0).</u> (the case ($if_>$) is similar.) The situation is that

$$se \vdash \mathtt{if0}^l\ 0^{l_0}\ \mathtt{then}\ ue_1^{l_1}\ \mathtt{else}\ e_2 \Rightarrow ue_1^l.$$

By assumption we have $F \models^{me} \mathtt{if0}^l\ 0^{l_0}\ \mathtt{then}\ ue_1^{l_1}\ \mathtt{else}\ e_2$, implying $F \models^{me} ue_1^{l_1}$ and $\mathcal{C}_F(l_1, me) \leq_V \mathcal{C}_F(l, me)$. By Lemma B.4 this demonstrates that $F \models^{me} ue_1^l$, as desired.

<u>(*bind*).</u> The situation is that $se \vdash \mathtt{bind}^l\ se_1\ \mathtt{in}\ e_1 \Rightarrow \mathtt{bind}^l\ se_1\ \mathtt{in}\ e_1'$ because $se_1 \vdash e_1 \Rightarrow e_1'$, where there exists l_1, ue_1 and ue_1' such that $e_1 = ue_1^{l_1}$ and $e_1' = {ue_1'}^{l_1}$.

By assumption we have $F \models^{me} \mathtt{bind}^l\ se_1\ \mathtt{in}\ e_1$, that is there exists me_1 with $se_1\ \mathcal{R}_F\ me_1$ such that $F \models^{me_1} e_1$ and $\mathcal{C}_F(l_1, me_1) \leq_V \mathcal{C}_F(l, me)$. We can thus apply the induction hypothesis on the inference $se_1 \vdash e_1 \Rightarrow e_1'$, yielding $F \models^{me_1} e_1'$. This shows $F \models^{me} \mathtt{bind}^l\ se_1\ \mathtt{in}\ e_1'$, as desired.

<u>($bind_v$).</u> The situation is that $se \vdash \mathtt{bind}^l\ se_1\ \mathtt{in}\ sv^{l_1} \Rightarrow sv^l$. By our assumption $F \models^{me} \mathtt{bind}^l\ se_1\ \mathtt{in}\ sv^{l_1}$ we infer that there exists me_1 such that $\mathcal{C}_F(l_1, me_1) \leq_V \mathcal{C}_F(l, me)$ and such that $F \models^{me_1} sv^{l_1}$, which by Lemma B.3 amounts to $sv\ \mathcal{V}_F\ \mathcal{C}_F(l_1, me_1)$. By Lemma B.1 we infer $sv\ \mathcal{V}_F\ \mathcal{C}_F(l, me)$, which by Lemma B.3 amounts to the desired $F \models^{me} sv^l$.

<u>(app_v).</u> The situation is, with $sv_1 = \mathtt{close}\ fn_1\ \mathtt{in}\ se_1$ and $fn_1 = \mu f.\lambda x.e_0$ and $e_0 = ue_0^{l_0}$, that

$$se \vdash sv_1^{l_1} \mathbin{@_l} sv_2^{l_2} \Rightarrow \mathtt{bind}^l\ se_1[f \mapsto sv_1][x \mapsto sv_2]\ \mathtt{in}\ e_0.$$

By assumption we have

$$F \models^{me} sv_1^{l_1} \mathbin{@_l} sv_2^{l_2} \tag{2}$$

from which we infer that $F \models^{me} sv_1^{l_1}$ and $F \models^{me} sv_2^{l_2}$, which by Lemma B.3 amounts to

$$sv_1\ \mathcal{V}_F\ \mathcal{C}_F(l_1, me) \text{ and} \tag{3}$$
$$sv_2\ \mathcal{V}_F\ \mathcal{C}_F(l_2, me) \tag{4}$$

where (4) by Lemma B.2 implies that there exists v_2 such that

$$v_2 \in \mathcal{C}_F(l_2, me) \tag{5}$$
$$sv_2\ \mathcal{V}_F\ \{v_2\}. \tag{6}$$

From (3) we infer that there exists me_1 and M_1 such that $((fn_1, me_1), M_1) \in \mathcal{C}_F(l_1, me)$ and such that

$$se_1 \; \mathcal{R}_F \; me_1. \tag{7}$$

From (2) and (5) we therefore deduce, with $M = \Phi_F((l_2, me), (fn_1, me_1))$, that there exists $m \in M$ such that $\{v_2\} \leq_V \rho_F(x, m)$ which together with (6) by Lemma B.1 implies

$$sv_2 \; \mathcal{V}_F \; \rho_F(x, m). \tag{8}$$

For this m we deduce, still from (2), that

$$F \models^{me_1[f,x \mapsto m]} e_0 \tag{9}$$
$$\mathcal{C}_F(l_0, me_1[f, x \mapsto m]) \leq_V \mathcal{C}_F(l, me) \tag{10}$$
$$\{((fn_1, me_1), Mem_F)\} \leq_V \rho_F(f, m). \tag{11}$$

From (7) and (11) we deduce $sv_1 \; \mathcal{V}_F \; \rho_F(f, m)$ which together with (7) and (8) enables us to infer

$$se_1[f \mapsto sv_1][x \mapsto sv_2] \; \mathcal{R}_F \; me_1[f, x \mapsto m].$$

Together with (9) and (10), this amounts to $F \models^{me} \texttt{bind}^l \; se_1[f \mapsto sv_1][x \mapsto sv_2] \; \texttt{in} \; e_0$, as desired.

This completes the proof. □

B.2 Least Analysis

Lemma B.5. *Let P and M be given. Then there exists a flow analysis F for P such that F is valid and $Mem_F = M$. (Note that $MemEnv_F$, $FlowConf_F$ and $UnAnnFlowVal_F$ are determined by P and M). Moreover, we have:*

1. *if β is a mapping from $Labs_{\mathrm{e}}(P) \times MemEnv_F$ into Mem_F we can ensure that F belongs to $CallString^P_\beta$;*

2. *if α is a mapping from Mem_F into $\mathcal{P}(UnAnnFlowVal_F)$ with $Cover_\alpha$ we can ensure that F belongs to $ArgBased^\alpha_.$*

□

Proof. First define $X = \{(e, me) \in FlowConf_F \mid e \in SubExpr_P\}$. Next we define $\mathcal{C}_F$, ρ_F and Φ_F: for $(ue^l, me) \in X$ we stipulate $\mathcal{C}_F(l, me) = \iota_V(UnAnnFlowVal_F)$; for all f and m we stipulate $\rho_F(f, m) = \iota_V(UnAnnFlowVal_F)$; for all x and m we stipulate $\rho_F(x, m) = \iota_V(UnAnnFlowVal_F)$, except in case 2 where we stipulate $\rho_F(x, m) = \iota_V(\alpha(m))$; and for all l, me and ac we stipulate $\Phi_F((l, me), ac) = Mem_F$, except in case 1 where we stipulate $\Phi_F((l, me), ac) = \{\beta(l', me)\}$ with l' such that $e_1 \,@_{l'}\, ue_2^l \in SubExpr_P$.

Our task is to prove that F is valid, which amounts to showing that $F \models^{me} e$ for all $(e, me) \in X$. By the principle of coinduction, this can be done by demonstrating that $X \subseteq \mathcal{G}_F(X)$. So consider $(e, me) \in X$; we do a case analysis on e (which is pure) and in all cases we must establish $(e, me) \in \mathcal{G}_F(X)$.

$\underline{e = z^l.}$ Clearly $\bot \neq \rho_F(z, me(z)) \leq_V \mathcal{C}_F(l, me)$ (except for case 2, "$\leq_V$" is even "$=$"). This shows $(e, me) \in \mathcal{G}_F(X)$.

$\underline{e = \mu f.\lambda^l x.e_0.}$ Here $ac = (\mu f.\lambda x.e_0, me) \in AbsClos_F \subseteq UnAnnFlowVal_F$, and therefore $(ac, Mem_F) \in \iota_V(UnAnnFlowVal_F) = \mathcal{C}_F(l, me)$. This shows $(e, me) \in \mathcal{G}_F(X)$.

$\underline{e = e_1 \,@_l\, e_2.}$ Let $e_1 = ue_1^{l_1}$ and $e_2 = ue_2^{l_2}$. To demonstrate $(e, me) \in \mathcal{G}_F(X)$ we must first demonstrate $\mathcal{C}_F(l, me) \neq \bot$ and $(e_1, me) \in X$ and $(e_2, me) \in X$, which is obvious. Next we must consider a given $(ac_0, M_0) \in \mathcal{C}_F(l_1, me)$; let $ac_0 = (fn, me_0)$ with $fn = \mu f.\lambda x.e_0$ where $e_0 = ue_0^{l_0}$, and let $M' = \Phi_F((l_2, me), ac_0)$. Note that $M_0 = Mem_F$, that $fn \in Funs_{\mathrm{e}}(P)$ implying $e_0 \in SubExpr_P$, and that $FV_{\mathrm{e}}(fn) \subseteq dom(me_0)$. This clearly shows that $M' \subseteq M_0$ and that for all $m \in M'$ we have $(e_0, me_0[f, x \mapsto m]) \in X$, $\mathcal{C}_F(l_0, me_0[f, x \mapsto m]) \leq_V \mathcal{C}_F(l, me)$, $\rho_F(x, m) \neq \bot$, and $(ac_0, Mem_F) \in \iota_V(UnAnnFlowVal_F)$ implying $\{(ac_0, Mem_F)\} \leq_V \rho_F(f, m)$.

Thus the only task left is to verify that

$$\forall v \in \mathcal{C}_F(l_2, me).\ \exists m \in M'.\ \text{such that } \{v\} \leq_V \rho_F(x, m). \tag{12}$$

First observe that we can write v as $\iota_v(uv)$. In case 2, we have $Cover_\alpha$ and therefore there exists $m \in Mem_F$ such that $uv \in \alpha(m)$ implying $v \in \rho_F(x, m)$; since $M' = Mem_F$ this establishes (12). Otherwise, (12) holds trivially since M' is non-empty.

The remaining cases. They can easily be handled, using the techniques from the previous cases. □

Lemma B.6. *Let* $\{F_j \mid j \in J\}$ *be a* non-empty *family of flow analyses for* P*, such that* Mem_{F_j} *does not depend on* j *and such that for all* j *it holds that* F_j *is valid. Suppose that either*

1. *there exists* β *such that all* F_j *belong to* $CallString^P_\beta$*;*
2. *there exists* α *with* $Disj_\alpha$ *such that all* F_j *belong to* $ArgBased^P_\alpha$*.*

Let $F = \langle P, Mem_F, \mathcal{C}_F, \rho_F, \Phi_F\rangle$ *be defined as* $\sqcap_{j\in J} F_j$*, that is:* $Mem_F = Mem_{F_j}$ *for all* $j \in J$*; for all* l *and* me *it holds that* $\mathcal{C}_F(l, me) = \sqcap^V_{j\in J}\mathcal{C}_{F_j}(l, me)$*; for all* z *and* m *it holds that* $\rho_F(z, m) = \sqcap^V_{j\in J}\rho_{F_j}(z, m)$*; and for all* l*,* me *and* ac *it holds that* $\Phi_F((l, me), ac) = \cap_{j\in J}\Phi_{F_j}((l, me), ac)$ *(the left hand side is defined iff the right hand side is).*

Then F *is valid. In case 1,* F *will belong to* $CallString^P_\beta$*; in case 2,* F *will belong to* $CallString^P_\alpha$*.* □

Proof. First we show that F belongs to the appropriate categories. In the case 1, if $\Phi_F((l_2, me), ac)$ is defined then for all $j \in J$ also $\Phi_{F_j}((l_2, me), ac)$ is defined and therefore given by $\{\beta(l, me)\}$ (where l is such that $e_1 \,@_l\, ue_2^{l_2} \in SubExpr_P$); thus $\Phi_F((l_2, me), ac) = \{\beta(l, me)\}$ as desired. In the case 2, if $\rho_F(x, m) \neq \bot$ then for all $j \in J$ also $\rho_{F_j}(x, m) \neq \bot$ and therefore $\epsilon_V(\rho_{F_j}(x, m)) = \alpha(m)$; this clearly implies the desired relation $\epsilon_V(\rho_F(x, m)) = \alpha(m)$.

Now to the main obligation of proving that F is valid. For that purpose we define

$$X = \{(e, me) \in FlowConf_F \mid e \in SubExpr_P \text{ and } \forall j \in J.\, F_j \models^{me} e\}.$$

Exploiting that for all $j \in J$ it holds that F_j is valid, we infer that $(P, \epsilon) \in X$, and that if $\mathcal{C}_F(l, me) \neq \bot$ with $e = ue^l \in SubExpr_P$ and $(e, me) \in FlowConf_F$ then for all $j \in J$ it holds that $\mathcal{C}_{F_j}(l, me) \neq \bot$ and hence $F_j \models^{me} e$, that is $(e, me) \in X$. Our task is thus to show that $F \models^{me} e$ for all $(e, me) \in X$, which by the principle of coinduction can be accomplished by proving

$$X \subseteq \mathcal{G}_F(X).$$

So consider $(e, me) \in X$; we do a case analysis on e (which is pure) and in all cases we must establish $(e, me) \in \mathcal{G}_F(X)$.

$e = z^l$. For all $j \in J$ we have $F_j \models^{me} z^l$, that is $\bot \neq \rho_{F_j}(z, me(z)) \leq_V \mathcal{C}_{F_j}(l, me)$. Then clearly $\bot \neq \rho_F(z, me(z)) \leq_V \mathcal{C}_F(l, me)$, showing $(e, me) \in \mathcal{G}_F(X)$ as desired.

$e = \mu f.\lambda^l x.e_0$. For all $j \in J$ we have $F_j \models^{me} e$, that is $\left\{((\mu f.\lambda x.e_0, me), Mem_{F_j})\right\} \leq_V \mathcal{C}_{F_j}(l, me)$ where $Mem_{F_j} = Mem_F$. Then clearly $\{((\mu f.\lambda x.e_0, me), Mem_F)\} \leq_V \mathcal{C}_F(l, me)$, showing $(e, me) \in \mathcal{G}_F(X)$ as desired.

$e = e_1 \,@_l\, e_2$. Let $e_1 = ue_1^{l_1}$ and $e_2 = ue_2^{l_2}$. By assumption we have

$$\forall j \in J.\ F_j \models^{me} e \tag{13}$$

from which we deduce

$$\forall j \in J.\ F_j \models^{me} e_1 \text{ and } F_j \models^{me} e_2 \text{ and } \mathcal{C}_{F_j}(l, me) \neq \bot$$

which (since for $i = 1, 2$ we have $(e_i, me) \in \mathit{FlowConf}_F$ and $e_i \in \mathit{SubExpr}_P$) shows that

$$(e_1, me) \in X \text{ and } (e_2, me) \in X \text{ and } \mathcal{C}_F(l, me) \neq \bot. \tag{14}$$

Now assume $(ac_0, M_0) \in \mathcal{C}_F(l_1, me)$, where $ac_0 = (\mu f.\lambda x.e_0, me_0)$ with $e_0 = ue_0^{l_0}$. Clearly

$$e_0 \in \mathit{SubExpr}_P \text{ and } \forall m \in \mathit{Mem}_F.\ \mathit{FV}_{\mathrm{e}}(e_0) \subseteq \mathit{dom}(me_0[f, x \mapsto m]). \tag{15}$$

For all $j \in J$ it holds that $\mathcal{C}_F(l_1, me) \leq_V \mathcal{C}_{F_j}(l_1, me)$, showing that for all $j \in J$ there exists M'_j with $M'_j \subseteq M_0$ such that $(ac_0, M'_j) \in \mathcal{C}_{F_j}(l_1, me)$. This, together with (13), implies that with $M_j = \Phi_{F_j}((l_2, me), ac_0)$ we have

$$\forall j \in J.\ M_j \subseteq M'_j \subseteq M_0 \tag{16}$$

$$\forall j \in J. \forall v \in \mathcal{C}_{F_j}(l_2, me).\ \exists m \in M_j.\ \{v\} \leq_V \rho_{F_j}(x, m) \tag{17}$$

$$\begin{aligned} \forall j \in J. \forall m \in M_j.\, & F_j \models^{me_0[f,x\mapsto m]} e_0 \\ & \mathcal{C}_{F_j}(l_0, me_0[f, x \mapsto m]) \leq_V \mathcal{C}_{F_j}(l, me) \\ & \rho_{F_j}(x, m) \neq \bot \text{ and } \{(ac_0, \mathit{Mem}_F)\} \leq_V \rho_{F_j}(f, m) \end{aligned} \tag{18}$$

Let $M = \Phi_F((l_2, me), ac_0)$; note that M is defined and equals $\cap_{j \in J} M_j$. Using (16) we infer

$$M \subseteq M_0 \tag{19}$$

and using (18) and (15) it is easy to see that

$$\begin{aligned} \forall m \in M.\, & (e_0, me_0[f, x \mapsto m]) \in X \\ & \mathcal{C}_F(l_0, me_0[f, x \mapsto m]) \leq_V \mathcal{C}_F(l, me) \\ & \rho_F(x, m) \neq \bot \text{ and } \{(ac_0, \mathit{Mem}_F)\} \leq_V \rho_F(f, m) \end{aligned} \tag{20}$$

Note that (14), (19), and (20) almost demonstrates the desired relation $(e, me) \in \mathcal{G}_F(X)$; we only need to establish

$$\forall v \in \mathcal{C}_F(l_2, me). \exists m \in M.\ \{v\} \leq_V \rho_F(x, m).$$

So let $v \in \mathcal{C}_F(l_2, me)$ be given; our goal is to find $m \in M$ such that

$$\{v\} \leq_V \rho_F(x, m). \tag{21}$$

For all $j \in J$ there exists $v_j \in \mathcal{C}_{F_j}(l_2, me)$ with $v \leq_v v_j$; by (17) this implies that there exists $m_j \in M_j$ such that

$$\{v_j\} \leq_V \rho_{F_j}(x, m_j). \tag{22}$$

It will be sufficient to show that

$$\exists m. \forall j \in J.\, m_j = m. \tag{23}$$

For then we from (22) infer that for all $j \in J$ it holds that $\{v\} \leq_V \rho_{F_j}(x, m)$, clearly establishing (21).

We now split the analysis, according to the cases listed in the assumption of the lemma. In case 1, we for all $j \in J$ have $M = M_j = \{\beta(l, me)\}$ which trivially implies (23).

Next we address case 2, where for all $j \in J$ it holds that $F_j \in \mathit{ArgBased}_\alpha^P$ with Disj_α. This establishes (23), since by (22)

$$\forall j \in J.\ \epsilon_v(v) = \epsilon_v(v_j) \in \epsilon_V(\rho_{F_j}(x, m_j)) = \alpha(m_j).$$

The remaining cases. They can easily be handled, using the techniques from the previous cases.

□

B.3 Connection to P&P

Proof of Lemma 4.12. We define

$$X = \big\{\{(e, me) \in \mathit{FlowConf}_F \text{ with } e \in \mathit{SubExpr}_P \mid \exists s.\, (\gamma_{me}^{-1}(me), e, s) \in R\}\big\}$$

and our task can be accomplished (as $(P, \epsilon) \in X$ follows from R being an F-analysis) by showing that for all $(e, me) \in X$ we have $F \models^{me} e$. By the principle of coinduction this can be done by showing $X \subseteq \mathcal{G}_F(X)$. So consider $(e, me) \in X$ with $e = ue^l$; we do a case analysis on ue (which is pure). In all cases we let $\rho = \gamma_{me}^{-1}(me)$ and let $S = \{\{s \mid (\rho, e, s) \in R\}\}$ (which is non-empty), and must establish $(e, me) \in \mathcal{G}_F(X)$.

$ue = x$. Let $m = me(x)$. For all $s \in S$ we have $\gamma^{-1}(m) = \rho(x) \subseteq s$, and thus $\gamma^{-1}(m) \subseteq \cap S$. This clearly implies the desired relation

$$\perp \neq \rho_F(x, m) = \gamma_V(\gamma^{-1}(m)) \leq_V \gamma_V(\cap S) = \mathcal{C}_F(l, me).$$

$ue = \lambda x.e_0$. For all $s \in S$ we have $(\lambda x.e_0, \rho) \in s$, and thus $(\lambda x.e_0, \rho) \in \cap S$. This implies the desired relation

$$\{((\lambda x.e_0, me), \mathit{Mem}_F)\} = \{\gamma_v((\lambda x.e_0, \rho))\} \leq_V \gamma_V(\cap S) = \mathcal{C}_F(l, me).$$

$ue = e_1 \,@\, e_2$. For $i = 1, 2$ we let $e_i = ue_i^{l_i}$, and define $S_i = \{s \mid (\rho, e_i, s) \in R\}$.

Our first task is to show that $(e_1, me) \in X$ and $(e_2, me) \in X$ which amounts to S_1 and S_2 being non-empty, but this follows from S being non-empty.

Next we must consider $(ac_0, M_0) \in \mathcal{C}_F(l_1, me)$, with $ac_0 = (\lambda x.e_0, me_0)$ and $e_0 = ue_0^{l_0}$. Let $\rho_0 = \gamma_{me}^{-1}(me_0)$. We infer that $M_0 = \mathit{Mem}_F$, and that $(\lambda x.e_0, \rho_0) \in \cap S_1$. We now exploit our assumption that R has the deterministic cache property, and define $C = \mathit{Cach}_R((e_1 \,@_l\, e_2, \rho), (\lambda x.e_0, \rho_0))$. Let $M = \Phi_F((l_2, me), ac_0)$, that is $M = \{\gamma(s') \mid s' \in C\}$. Trivially $M \subseteq M_0$.

For the rest of the proof, we assume that some $s \in S$ is given. Since R is an F-analysis there exists $s_1 \in S_1$ and $s_2 \in S_2$ such that $s_2 \subseteq \cup C$ and, since $(\lambda x.e_0, \rho_0) \in \cap S_1 \subseteq s_1$, for all $s' \in C$ there exists s'' such that $(\rho_0[x \mapsto s'], e_0, s'') \in R$ and $s'' \subseteq s$.

Next we consider $v \in \mathcal{C}_F(l_2, me) = \gamma_V(\cap S_2) \subseteq \gamma_V(s_2)$. From $s_2 \subseteq \cup C$ we infer that there exists $s' \in C$ such that $v \in \gamma_V(s')$. With $m = \gamma(s')$ we have the desired relation $v \in \gamma_V(\gamma^{-1}(m)) = \rho_F(x, m)$.

Finally, we must consider $m \in M$. (Trivially, $\rho_F(x, m) \neq \perp$.) Let

$$S' = \big\{s'' \mid (\rho_0[x \mapsto \gamma^{-1}(m)], e_0, s'') \in R\big\}.$$

From the above we infer (since $\gamma^{-1}(m) \in C$) that S' is not empty, implying (since $e_0 \in \mathit{SubExpr}_P$) that $(e_0, me_0[x \mapsto m]) \in X$, and that there exists $s'' \in S'$ with $s'' \subseteq s$. This shows that $\cap S' \subseteq s$, and as S' does not depend on our choice of s we infer that $\cap S' \subseteq \cap S$ which implies $\mathcal{C}_F(l_0, me_0[x \mapsto m]) \leq_V \mathcal{C}_F(l, me)$. This concludes the proof of this case.

$ue = c$. For all $s \in S$ we have $\mathrm{Int} \in s$, and thus $\mathrm{Int} \in \cap S$. This implies the desired relation $\mathrm{Int} \in \gamma_V(\cap S) = \mathcal{C}_F(l, me)$.

$ue = \texttt{succ}\ e_1$. Let s belong to S. Then there exists s_1 such that $(\rho, e_1, s_1) \in R$, showing that $(e_1, me) \in X$. Moreover, $\mathrm{Int} \in s$. Therefore $\mathrm{Int} \in \cap S$, implying $\mathrm{Int} \in \gamma_V(\cap S) = \mathcal{C}_F(l, me)$.

$ue = \texttt{if0}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2$. For $i = 0, 1, 2$ we let $e_i = ue_i^{l_i}$, and define $S_i = \{s \mid (\rho, e_i, s) \in R\}$. Since S is non-empty, each S_i is non-empty, and thus for $i = 0, 1, 2$ we have $(e_i, me) \in X$. For all $s \in S$ there exists $s_1 \in S_1$ and $s_2 \in S_2$ such that $s_1 \subseteq s$ and $s_2 \subseteq s$; this shows that for $i = 1, 2$ we have $\cap S_i \subseteq \cap S$ and therefore the desired relation

$\mathcal{C}_F(l_i, me) = \gamma_V(\cap S_i) \leq_V \gamma_V(\cap S) = \mathcal{C}_F(l, me).$

$\square$

Proof of Lemma 4.14. Since F is valid we have $F \models^{\epsilon} P$ so clearly there exists s such that $(\epsilon, P, s) \in R$.

To check the conditions in Fig. 11, assume that $(\rho, e, s) \in R$ because with $e = ue^l \in \mathit{SubExpr}_P$ we have $F \models^{me} e$ with $\rho = \delta_{me}(me)$ and $s = \delta_V(\mathcal{C}_F(l, me))$. We now perform case analysis on ue:

$\underline{ue = x.}$ Let $m = me(x)$. From $\rho_F(x, m) \leq_V \mathcal{C}_F(l, me)$ we infer the desired relation

$$\begin{array}{rcl}\rho(x) & = & \{\delta_{uv}(uv) \mid uv \in \alpha(m)\} = \{\delta_{uv}(uv) \mid uv \in \epsilon_V(\rho_F(x, m))\} \\ & \leq_V & \{\delta_{uv}(uv) \mid uv \in \epsilon_V(\mathcal{C}_F(l, me))\} = \delta_V(\mathcal{C}_F(l, me)) \\ & = & s.\end{array}$$

$\underline{ue = \lambda x.e_0.}$ From $\{((\lambda x.e_0, me), \mathit{Mem}_F)\} \leq_V \mathcal{C}_F(l, me)$ and by monotonicity of δ_V we obtain the desired relation $\{(\lambda x.e_0, \rho)\} \subseteq s$.

$\underline{ue = e_1 \mathbin{@} e_2.}$ For $i = 1, 2$ we let $e_i = ue_i^{l_i}$, and from $F \models^{me} e_i$ we infer that $\mathcal{C}_F(l_i, me) \neq \bot$ so with $s_i = \delta_V(\mathcal{C}_F(l_i, me))$ it holds that $(\rho, e_i, s_i) \in R$.

Now let $(\lambda x.e_0, \rho_0) \in s_1$ with $e_0 = ue_0^{l_0}$. There thus exists $((\lambda x.e_0, me_0), M_0) \in \mathcal{C}_F(l_1, me)$ with $\rho_0 = \delta_{me}(me_0)$. From $F \models^{me} e$ we infer that there exists M with $M \subseteq M_0$ such that for all $v \in \mathcal{C}_F(l_2, me)$ there exists $m \in M$ with $\{v\} \leq_V \rho_F(x, m)$ and such that for all $m \in M$ we have $F \models^{me_0[x \mapsto m]} e_0$ and $\mathcal{C}_F(l_0, me_0[x \mapsto m]) \leq_V \mathcal{C}_F(l, me)$. We then define

$$C = \{s_m \mid m \in M\} \text{ where } s_m = \{\delta_{uv}(uv) \mid uv \in \alpha(m)\}.$$

We must first prove that $s_2 \subseteq \cup C$. So let $a \in s_2$ be given. There exists $v \in \mathcal{C}_F(l_2, me)$ such that $a = \delta_{uv}(\epsilon_v(v))$. And from the above we infer that there exists $m \in M$ such that $\{v\} \leq_V \rho_F(x, m)$, implying that $\epsilon_v(v) \in \epsilon_V(\rho_F(x, m)) = \alpha(m)$ which shows the desired relation $a \in s_m \subseteq \cup C$.

Next let $s' \in C$ be given, with $s' = s_m$ for some $m \in M$. From $F \models^{me_0[x \mapsto m]} e_0$ and $\bot \neq \mathcal{C}_F(l_0, me_0[x \mapsto m]) \leq_V \mathcal{C}_F(l, me)$ we infer (using the monotonicity of δ_V) that with $s'' = \delta_V(\mathcal{C}_F(l_0, me_0[x \mapsto m]))$ we have $s'' \subseteq s$, and (since $\delta_{me}(me_0[x \mapsto m]) = \rho_0[x \mapsto s_m]$) we also infer that $(\rho_0[x \mapsto s'], e_0, s'') \in R$.

$\underline{ue = c.}$ From $\mathrm{Int} \in \mathcal{C}_F(l, me)$ we infer that $\mathrm{Int} \in s$.

$\underline{ue = \texttt{succ}\ e_1.}$ Let $e_1 = ue_1^{l_1}$. We know that $\mathrm{Int} \in \mathcal{C}_F(l, me)$, implying that $\mathrm{Int} \in s$, and that $F \models^{me} e_1$, implying that with $s_1 = \delta_V(\mathcal{C}_F(l_1, me))$ we have $(\rho, e_1, s_1) \in R$.

$\underline{ue = \texttt{if0}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2.}$ For $i = 0, 1, 2$ we let $e_i = ue_i^{l_i}$, and from $F \models^{me} e_i$ we infer that $\mathcal{C}_F(l_i, me) \neq \bot$ so with $s_i = \delta_V(\mathcal{C}_F(l_i, me))$ it holds that $(\rho, e_i, s_i) \in R$. Moreover, since for $i = 1, 2$ we have $\mathcal{C}_F(l_i, me) \leq_V \mathcal{C}_F(l, me)$ it holds by monotonicity of δ_V that $s_1 \subseteq s$ and $s_2 \subseteq s$.

$\square$

B.4 Reachability

Proof of Lemma 4.16. The last claim of the lemma is trivial. For the first claim, we proceed by induction in the "derivation" of $(ue^l, me) \in \mathit{Reach}_P^F$; let $e = ue^l$.

[prg]. Here $e = P$ and $me = \epsilon$. Since F is valid we have $F \models^{\epsilon} P$, implying *(i)*. *(ii)* is void.

[fun]. Here me takes the form $me'[x, f \mapsto m']$, and *(i)* is among the premises of [fun]. For *(ii)*, assume that $(z \mapsto m) \in me$. If $m = m'$ and $z \in \{x, f\}$, $(z, m) \in \mathit{Reach}_P^F$ is part of

the conclusion of [fun]. Otherwise, $(z \mapsto m) \in me'$ so $(z, m) \in \mathit{Reach}_P^F$ follows from the induction hypothesis applied to $(\mu f.\lambda x.e, me') \in \mathit{Reach}_P^F$.

[app]. Assume that the premise takes the form $(e \mathbin{@_{l'}} e_2, me) \in \mathit{Reach}_P^F$ (the symmetric case is similar). The induction hypothesis tells us that $\mathcal{C}_F(l', me) \neq \bot$, so as F is valid we infer $F \models^{me} e \mathbin{@_{l'}} e_2$. Therefore $F \models^{me} e$, implying $\mathcal{C}_F(l, me) \neq \bot$. *(ii)* follows trivially from the induction hypothesis.

The remaining cases. They are similar to [app]. $\square$

C Type to Flow

First an auxiliary result:

Lemma C.1. *Suppose that D_T contains at address ke a judgement $A \vdash \mu f.\lambda^l x.e : u$ that is derived by $[\mathsf{fun}]^{(q:t)}$. Then for all $k \in \mathit{dom}(t)$ it holds that $\mathit{Analyzes}_k^F(\mu f.\lambda x.e, ke)$.* $\square$

Proof. Let $k \in \mathit{dom}(t)$ be given. D_T contains at address $ke[f, x \mapsto k]$ a judgement of the form $A[f \mapsto u_k''][x \mapsto u_k] \vdash e : u_k'$ where the side condition for $[\mathsf{fun}]^{(q:t)}$ ensures that $\bigvee(q : t) \leq_u u_k''$. Clearly $((\mu f.\lambda x.e, ke), \mathit{dom}(t))$ belongs to $\mathcal{F}_T(\bigvee(q : t))$, so by Lemma 5.2 we infer

$$\{((\mu f.\lambda x.e, ke), \mathit{Mem}_F)\} \leq_V \mathcal{F}_T(\bigvee(q : t)) \leq_V \mathcal{F}_T(u_k'') = \mathcal{F}_T(A_T(f, k)) = \rho_F(f, k).$$

This yields the desired result, since clearly $\rho_F(x, k) \neq \bot$ and (with $e = ue^l$) $\mathcal{C}_F(l, ke[f, x \mapsto k]) \neq \bot$. $\square$

Lemma C.2. *It holds that F is valid.* $\square$

Proof. We define

$$X = \{\{(e, me) \mid D_T \text{ contains a judgement for } e \text{ at address } me\}\}$$

Note that if $e = ue^l \in \mathit{SubExpr}_P$ and $\mathcal{C}_F(l, me) \neq \bot$ then $(e, me) \in X$; our task can thus be accomplished (as trivially $(P, \epsilon) \in X$) by showing that for all $(e, me) \in X$ we have $F \models^{me} e$ and by the principle of coinduction this can be done by showing $X \subseteq \mathcal{G}_F(X)$. So consider $(e, me) \in X$; we do a case analysis on $e = ue^l$ (which is pure) and in all cases we must establish $(e, me) \in \mathcal{G}_F(X)$, using the assumption that D_T contains at address me a judgement J of the form $A \vdash e : u$ (so $\mathcal{C}_F(l, me) = \mathcal{F}_T(u)$).

$e = z^l$. We have $A(z) = A_T(z, me(z))$ and $A(z) \leq_u u$, so by Lemma 5.2 we infer

$$\bot \neq \rho_F(z, me(z)) = \mathcal{F}_T(A_T(z, me(z))) \leq_V \mathcal{F}_T(u) = \mathcal{C}_F(l, me)$$

which shows $(e, me) \in \mathcal{G}_F(X)$.

$e = \mu f.\lambda^l x.e_0$. Let J be derived using $[\mathsf{fun}]^{(q:t)}$; then $\bigvee(q : t) \leq_u u$. With $ac = (\mu f.\lambda x.e_0, me)$ we infer that $(ac, \mathit{dom}(t)) \in \mathcal{F}_T(\bigvee(q : t))$, so Lemma 5.2 yields

$$\{(ac, \mathit{Mem}_F)\} \leq_V \mathcal{F}_T(\bigvee(q : t)) \leq_V \mathcal{F}_T(u) = \mathcal{C}_F(l, me)$$

which shows $(e, me) \in \mathcal{G}_F(X)$.

$e = e_1 \mathbin{@_l} e_2$. Let $e_1 = ue_1^{l_1}$ and $e_2 = ue_2^{l_2}$. Let $w^@$ be such that J is derived using $[\mathsf{app}]^{w^@}$; the premises of J take the form $A \vdash e_1 : u_1$ and $A \vdash e_2 : u_2$ where for all $q \in \mathit{dom}(u_1)$ we have

$$u_1.q \leq_t \bigwedge(w^@(q) : u_2 \to u). \tag{1}$$

We first observe that

$$(e_1, me) \in X \text{ and } (e_2, me) \in X \text{ and } \mathcal{C}_F(l, me) \neq \bot. \tag{2}$$

Now consider $(ac_0, M_0) \in \mathcal{C}_F(l_1, me) = \mathcal{F}_T(u_1)$, where $ac_0 = (\mu f.\lambda x.e_0, me_0)$ with $e_0 = ue_0^{l_0}$. Thus D_T at address me_0 contains a judgement J_0 of the form $A_0 \vdash \mu f.\lambda x.e_0 : u_0$ derived using $[\mathsf{fun}]^{(q_0:t_0)}$, with $q_0 \in dom(u_1)$ and

$$t_0 \leq_t u_1.q_0 \tag{3}$$

and with $M_0 = dom(u_1.q_0)$. We infer that with $M = w^{@}(q_0)$ we have

$$\Phi_F((l_2, me), ac_0) = M \tag{4}$$

and (1) tells us that

$$M \subseteq M_0. \tag{5}$$

t_0 takes the form $\bigwedge_{k\in K}\{\{k\} : u_k \to u'_k\}$; by (1) and (3) we infer that

$$\bigwedge_{k\in K}\{\{k\} : u_k \to u'_k\} \leq_t \bigwedge(M : u_2 \to u) \tag{6}$$

implying $M \subseteq K$.

Now let $m \in M$ be given. Clearly D_T will contain at address $me_0[f, x \mapsto m]$ (as a premise of J_0) a judgement of the form

$$A_0[f \mapsto u''_m][x \mapsto u_m] \vdash e_0 : u'_m.$$

We can now assert

$$(e_0, me_0[f, x \mapsto m]) \in X \tag{7}$$
$$\mathcal{C}_F(l_0, me[f, x \mapsto m]) = \mathcal{F}_T(u'_m) \leq_V \mathcal{F}_T(u) \leq_V \mathcal{C}_F(l, me) \tag{8}$$
$$\rho_F(x, m) \neq \perp \tag{9}$$
$$\{(ac_0, Mem_F)\} \leq_V \rho_F(f, m) \tag{10}$$

where (8) follows from Lemma 5.2 since (6) implies $u'_m \leq_u u$, and where (10) is provided by Lemma C.1.

(2), (4), (5), (7), (8), (9) and (10) almost establishes $(e, me) \in \mathcal{G}_F(X)$; the only task left is to prove

$$\forall v \in \mathcal{C}_F(l_2, me).\, \exists m \in M.\, \{v\} \leq_V \rho_F(x, m).$$

So let $v \in \mathcal{C}_F(l_2, me) = \mathcal{F}_T(u_2)$ be given; clearly there exists $q \in dom(u_2)$ such that $v \in \mathcal{F}_T(\bigvee(q : u_2.q))$. From (6) we then infer that there exists $m \in M$ such that $u_2.q \leq_t u_m.q$, which by Lemma 5.2 implies the desired relation

$$\{v\} \leq_V \mathcal{F}_T(\bigvee(q : u_2.q)) \leq_V \mathcal{F}_T(u_m) = \mathcal{F}_T(A_T(x, m)) = \rho_F(x, m).$$

<u>$e = c^l$.</u> Since $u_{\text{int}} \leq_u u$ we by Lemmas 5.2 and 5.3 infer that

$$\{\text{Int}\} = \mathcal{F}_T(u_{\text{int}}) \leq_V \mathcal{F}_T(u) = \mathcal{C}_F(l, me)$$

which shows $(e, me) \in \mathcal{G}_F(X)$.

<u>$e = \mathtt{succ}^l\, e_1$.</u> Since $u_{\text{int}} \leq_u u$ we by Lemmas 5.2 and 5.3 infer that $\{\text{Int}\} = \mathcal{F}_T(u_{\text{int}}) \leq_V \mathcal{F}_T(u) = \mathcal{C}_F(l, me)$. Together with $(e_1, me) \in X$ this shows $(e, me) \in \mathcal{G}_F(X)$.

<u>$e = \mathtt{if0}^l\, e_0\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2$.</u> Let $e_1 = ue_1^{l_1}$ and $e_2 = ue_2^{l_2}$. The premises of J (all at address me) take the form

$$A \vdash e_0 : u_0 \text{ and } A \vdash e_1 : u_1 \text{ and } A \vdash e_2 : u_2$$

where $u_1 \leq_u u$ and $u_2 \leq_u u$; so by Lemma 5.2 we infer that

$$\mathcal{C}_F(l_1, me) = \mathcal{F}_T(u_1) \leq_V \mathcal{F}_T(u) \leq_V \mathcal{C}_F(l, me) \text{ and } \mathcal{C}_F(l_2, me) = \mathcal{F}_T(u_2) \leq_V \mathcal{F}_T(u) \leq_V \mathcal{C}_F(l, me).$$

This establishes $(e, me) \in \mathcal{G}_F(X)$, since the above judgements show that $(e_0, me) \in X$ and $(e_1, me) \in X$ and $(e_2, me) \in X$. $\square$

Lemma C.3. *It holds that F is safe.* $\square$

Proof. Let $ue^l \in SubExpr_P$; we must consider several cases.

$\underline{ue = ue_1^{l_1} @ e_2.}$ Assume that $\mathcal{C}_F(l_1, me) \neq \perp$; we must show that Int does not belong to $\mathcal{C}_F(l_1, me)$. The situation is that there exists u_1 with $\mathcal{F}_T(u_1) = \mathcal{C}_F(l_1, me)$ such that $A \vdash ue_1^{l_1} : u_1$ occurs with address me in D_T, as a left premise of a judgement derived by $[\mathsf{app}]^{w^@}$. If $\mathrm{Int} \in \mathcal{C}_F(l_1, me)$ then $q_{\mathrm{int}} \in dom(u_1)$, but by the side condition of applying $[\mathsf{app}]^{w^@}$ this is impossible.

$\underline{ue = \texttt{succ}\ ue_1^{l_1}.}$ Assume that $\mathcal{C}_F(l_1, me) \neq \perp$. The situation is that there exists u_1 with $\mathcal{F}_T(u_1) = \mathcal{C}_F(l_1, me)$ such that $A \vdash ue_1^{l_1} : u_1$ occurs with address me in D_T, as the premise of a judgement derived by $[\mathsf{suc}]$. Therefore $u_1 \leq_u u_{\mathrm{int}}$, which by Lemmas 5.2 and 5.3 implies that $\mathcal{F}_T(u_1) \leq_V \mathcal{F}_T(u_{\mathrm{int}}) = \{\mathrm{Int}\}$. This shows that if $v \in \mathcal{C}_F(l_1, me)$ then $v = \mathrm{Int}$.

$\underline{ue = \texttt{if0}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2.}$ This case is similar to the previous case. $\square$

Lemma C.4. *Let $e = ue^l \in SubExpr_P$. If $\mathcal{C}_F(l, me) \neq \perp$ then $(e, me) \in Reach_P^F$.* $\square$

Proof. Our assumption is that D_T contains a judgement J for e at address me. We proceed by induction in the distance from J to the root of D_T. If J is in fact the root of D_T, then $e = P$ and $me = \epsilon$ so the claim is clear.

Otherwise we can assume that there exists e_0 and a judgement J_0 for e_0 that occurs at address me_0 such that J is a premise of J_0; the induction hypothesis tells us that $(e_0, me_0) \in Reach_P^F$. Therefore $(e, me) \in Reach_P^F$ follows immediately, unless e_0 is of the form $\mu f.\lambda^l x.e$ in which case there exists m such that $me = me_0[f, x \mapsto m]$. But also in this case we have $(e, me) \in Reach_P^F$, thanks to Lemma C.1. $\square$

Lemma C.5. *If $\rho_F(z, m) \neq \perp$ then $(z, m) \in Reach_P^F$.* $\square$

Proof. It is easy to see that our assumption ensures that there exists address me and $\mu f.\lambda x.e^l \in SubExpr_P$ with $z \in \{f, x\}$ such that $\mathcal{C}_F(l, me[f, x \mapsto m]) \neq \perp$. Lemma C.4 tells us that $(ue^l, me[f, x \mapsto m]) \in Reach_P^F$; inspecting the definition of $Reach_P^F$ then shows that also $(z, m) \in Reach_P^F$ must hold. $\square$

Lemma C.6. *If $(ue^l, me) \in Reach_P^F$ then $\mathcal{C}_F(l, me) \neq \perp$, and if $(z, m) \in Reach_P^F$ then $\rho_F(z, m) \neq \perp$.* $\square$

Proof. From Lemma C.2 we know that F is valid, so the claim follows from Lemma 4.16. $\square$

D Flow to Type

Proof of Theorem 6.6. We are given $e = ue^l$ and me such that $(e, me) \in Reach_P^F$, and we want to show that the judgement $J =$

$$\mathcal{T}_F^A(me) \vdash e : \mathcal{T}_F(\mathcal{C}_F(l, me))$$

(which has address me) is derivable from its premises. We do a case analysis on ue (which is pure); in all cases we employ that since $\mathcal{C}_F(l, me) \neq \perp$ (by Lemma 4.16) then (as F is valid) it holds that $F \models^{me} e$.

$\underline{ue = z.}$ Since $F \models^{me} e$ we have $\perp \neq \rho_F(z, me(z)) \leq_V \mathcal{C}_F(l, me)$ which by Lemma 6.2 implies

$$\mathcal{T}_F^A(me)(z) = \mathcal{T}_F^\rho(z, me(z)) \leq_u \mathcal{T}_F(\mathcal{C}_F(l, me)).$$

This shows that J is derivable from its premises (of which there are none).

$\underline{ue = \mu f.\lambda x.e_0.}$ Let $e_0 = ue_0^{l_0}$, let $ac = (\mu f.\lambda x.e_0, me)$, let $v = (ac, Mem_F)$, and let $M = \Big\{m \mid (e_0, me[f, x \mapsto m]) \in Reach_P^F\Big\}$. Then the set of premises of J can be written as

$$\big\{\mathcal{T}_F^A(me[f,x \mapsto m]) \vdash e_0 : \mathcal{T}_F(\mathcal{C}_F(l_0, me[f,x \mapsto m])) \mid m \in M\big\}$$

with

$$\mathcal{T}_F^A(me[f,x \mapsto m]) = \mathcal{T}_F^A(me)[f \mapsto \mathcal{T}_F(\rho_F(f,m))][x \mapsto \mathcal{T}_F(\rho_F(x,m))].$$

Note that $M = \Big\{m \mid Analyzes_m^F(ac)\Big\}$ (so $m \in M$ implies $\{v\} \leq_V \rho_F(f,m)$) which shows that

$$\mathcal{T}_F(\{v\}) = \bigvee(ac : \textstyle\bigwedge_{m \in M}\{\{m\} : \mathcal{T}_F(\rho_F(x,m)) \to \mathcal{T}_F(\mathcal{C}_F(l_0, me[f,x \mapsto m]))\}).$$

Since $F \models^{me} e$ we have $\{v\} \leq_V \mathcal{C}_F(l, me)$, so Lemma 6.2 tells us that

$$\mathcal{T}_F(\{v\}) \leq_u \mathcal{T}_F(\mathcal{C}_F(l, me)) \text{ and } \forall m \in M.\, \mathcal{T}_F(\{v\}) \leq_u \mathcal{T}_F(\rho_F(f,m)).$$

This demonstrates that J is derivable from its premises by $[\mathsf{fun}]^{w^\lambda}$ with $w^\lambda = (ac : \mathcal{T}_F(\{v\}).ac)$.

$\underline{ue = e_1 \mathbin{@} e_2.}$ Let $e_1 = ue_1^{l_1}$ and $e_2 = ue_2^{l_2}$. The premises of J take the form

$$\mathcal{T}_F^A(me) \vdash e_1 : \mathcal{T}_F(\mathcal{C}_F(l_1, me)) \text{ and } \mathcal{T}_F^A(me) \vdash e_2 : \mathcal{T}_F(\mathcal{C}_F(l_2, me)).$$

Let $q_0 \in dom(\mathcal{T}_F(\mathcal{C}_F(l_1, me)))$ be given. In order to show that J is derived from its premises using $[\mathsf{app}]^{w^@}$ where $w^@$ is as in the theorem, we must show

$$\begin{array}{ll} & \mathcal{T}_F(\mathcal{C}_F(l_1, me)).q_0 \\ \leq_t & \bigwedge(\Phi_F((l_2, me), q_0) : \mathcal{T}_F(\mathcal{C}_F(l_2, me)) \to \mathcal{T}_F(\mathcal{C}_F(l, me))). \end{array} \tag{1}$$

Since F is safe, Int is not in $\mathcal{C}_F(l_1, me)$ so we infer that there exists $ac_0 = (\mu f.\lambda x.e_0, me_0)$ with $e_0 = ue_0^{l_0}$ such that $q_0 = ac_0$, and that there exists M_1 such that $(ac_0, M_1) \in \mathcal{C}_F(l_1, me)$. Then

$$\begin{array}{l} \mathcal{T}_F(\mathcal{C}_F(l_1, me)).q_0 = \bigwedge_{m \in M_0}\{\{m\} : \mathcal{T}_F(\rho_F(x,m)) \to \mathcal{T}_F(\mathcal{C}_F(l_0, me_0[f,x \mapsto m]))\} \\ \qquad\qquad \text{where } M_0 = \Big\{m \in M_1 \mid Analyzes_m^F(ac_0)\Big\}. \end{array}$$

From $F \models^{me} e$ and $(ac_0, M_1) \in \mathcal{C}_F(l_1, me)$ we infer that $M = \Phi_F((l_2, me), ac_0)$ is well-defined, that $M \subseteq M_1$ and subsequently that $M \subseteq M_0$. To demonstrate (1) we still need to establish

$$\forall m \in M.\, \mathcal{T}_F(\mathcal{C}_F(l_0, me_0[f,x \mapsto m])) \leq_u \mathcal{T}_F(\mathcal{C}_F(l, me)) \tag{2}$$
$$\forall q \in dom(\mathcal{T}_F(\mathcal{C}_F(l_2, me))).\, \exists m \in M.\, \mathcal{T}_F(\mathcal{C}_F(l_2, me)).q \leq_t \mathcal{T}_F(\rho_F(x,m)).q. \tag{3}$$

Concerning (2), we infer for $m \in M$ from $F \models^{me} e$ that $\mathcal{C}_F(l_0, me_0[f,x \mapsto m]) \leq_V \mathcal{C}_F(l, me)$ which by Lemma 6.2 implies the desired relation.

Concerning (3), we for $q \in dom(\mathcal{T}_F(\mathcal{C}_F(l_2, me)))$ infer that there exists $v \in \mathcal{C}_F(l_2, me)$ such that $\mathcal{T}_F(\{v\}).q = \mathcal{T}_F(\mathcal{C}_F(l_2, me)).q$. Still employing $F \models^{me} e$, we then see that there exists $m \in M$ such that $\{v\} \leq_V \rho_F(x,m)$ which by Lemma 6.2 implies $\mathcal{T}_F(\{v\}) \leq_u \mathcal{T}_F(\rho_F(x,m))$ and hence the desired relation.

$\underline{ue = c.}$ From $F \models^{me} e$ we infer $\{\mathrm{Int}\} \leq_V \mathcal{C}_F(l, me)$ which by Lemmas 6.2 and 6.3 implies $u_{\mathrm{int}} \leq_u \mathcal{T}_F(\mathcal{C}_F(l, me))$, showing that J is derivable from its (empty) set of premises.

$\underline{ue = \texttt{succ}\, e_1.}$ Let $e_1 = ue_1^{l_1}$. The premise of J takes the form

$$\mathcal{T}_F^A(me) \vdash e_1 : \mathcal{T}_F(\mathcal{C}_F(l_1, me)).$$

From $F \models^{me} e$ we deduce $\{\mathrm{Int}\} \leq_V \mathcal{C}_F(l, me)$ and since F is safe we have $\mathcal{C}_F(l_1, me) \leq_V \{\mathrm{Int}\}$. By Lemmas 6.2 and 6.3 this implies

$$\mathcal{T}_F(\mathcal{C}_F(l_1, me)) \leq_u u_{\mathrm{int}} \leq_u \mathcal{T}_F(\mathcal{C}_F(l, me))$$

which shows that J is derivable from its premises.

$\underline{ue = \texttt{if0}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2.}$ Let $e_i = ue_i^{l_i}$ for $i \in \{0, 1, 2\}$. The premises of J take the form

$\mathcal{T}_F^A(me) \vdash e_0 : \mathcal{T}_F(\mathcal{C}_F(l_0, me))$ and $\mathcal{T}_F^A(me) \vdash e_1 : \mathcal{T}_F(\mathcal{C}_F(l_1, me))$ and $\mathcal{T}_F^A(me) \vdash e_2 : \mathcal{T}_F(\mathcal{C}_F(l_2, me))$.

From $F \models^{me} e$ we deduce $\mathcal{C}_F(l_1, me) \leq_V \mathcal{C}_F(l, me)$ and $\mathcal{C}_F(l_2, me) \leq_V \mathcal{C}_F(l, me)$, and since F is safe we have $\mathcal{C}_F(l_0, me) \leq_V \{\mathrm{Int}\}$. By Lemmas 6.2 and 6.3 this implies

$\mathcal{T}_F(\mathcal{C}_F(l_1, me)) \leq_u \mathcal{T}_F(\mathcal{C}_F(l, me))$ and $\mathcal{T}_F(\mathcal{C}_F(l_2, me)) \leq_u \mathcal{T}_F(\mathcal{C}_F(l, me))$ and $\mathcal{T}_F(\mathcal{C}_F(l_0, me)) \leq_u u_{\mathrm{int}}$

which shows that J is derivable from its premises.

The last claim, concerning uniformity, is obvious except that we must show that whenever $\mathcal{T}_F^\rho(z, m)$ is defined then D_T in fact contains a judgement $A \vdash e : u$ with address me such that $me(z) = m$. But for such z and m we have $(z, m) \in \mathit{Reach}_P^F$ so there exists $\mu f.\lambda x.e$ and me with $z \in \{f, x\}$ such that $(e, me[f, x \mapsto m]) \in \mathit{Reach}_P^F$; this shows that D_T in fact does contain a judgement with address $me[f, x \mapsto m]$. □

E Round Trips

Proof of Theorem 7.2. First note that $T = \mathcal{T}_F$ is uniform (by Theorem 6.6), so $F' = \mathcal{F}(\mathcal{T}(F))$ is valid and safe (by Theorem 5.6). Clearly $\mathit{Mem}_{F'} = \mathit{IT}_T = \mathit{Mem}_F$.

For the claim that $\mathit{Reach}_P^{F'} = \mathit{Reach}_P^F$, we observe (by Theorems 5.6 and 6.6 and by Definitions 5.4 and 6.4) that $(e, me) \in \mathit{Reach}_P^{F'}$ iff (with $e = ue^l$) $\mathcal{C}_{F'}(l, me) \neq \perp$ iff D_T contains a judgement for e with address me iff $(e, me) \in \mathit{Reach}_P^F$, and that $(z, m) \in \mathit{Reach}_P^{F'}$ iff $\rho_{F'}(z, m) \neq \perp$ iff $A_T(z, m)$ is defined iff $\mathcal{T}_F^\rho(z, m)$ is defined iff $(z, m) \in \mathit{Reach}_P^F$.

We next establish an auxiliary result:

Lemma E.1. *For $V \in \mathit{FlowSet}_F$ it holds that $\mathcal{F}_T(\mathcal{T}_F(V)) = \mathrm{filter}_P^F(V)$.* □

Proof. Clearly $\mathrm{Int} \in \mathcal{F}_T(\mathcal{T}_F(V))$ iff $q_{\mathrm{int}} \in \mathit{dom}(\mathcal{T}_F(V))$ iff $\mathrm{Int} \in V$. In the following, let $ac = (\mu f.\lambda x.e, me)$ with $e = ue^l$.

First assume that $(ac, M') \in \mathcal{F}_T(\mathcal{T}_F(V))$. There thus exists $q \in \mathit{dom}(\mathcal{T}_F(V))$ such that $M' = \mathit{dom}(\mathcal{T}_F(V).q)$ and such that a judgement for $\mu f.\lambda x.e$ occurs in D_T with address me and is derived by $[\mathsf{fun}]^{w^\lambda}$ where w^λ takes the form $(q : t)$; by Theorem 6.6 we infer that $q = ac$. Since $ac \in \mathit{dom}(\mathcal{T}_F(V))$ we next infer that there exists M such that $(ac, M) \in V$and such that $M' = \mathit{dom}(\mathcal{T}_F(V).ac)$ is given by

$$\left\{m \in M \mid \mathit{Analyzes}_m^F(ac)\right\}$$

which since $(\mu f.\lambda x.e, me) \in \mathit{Reach}_P^F$ (by Definition 6.4) implies that

$$M' = \left\{m \in M \mid (e, me[f, x \mapsto m]) \in \mathit{Reach}_P^F\right\}.$$

This shows $(ac, M') \in \mathrm{filter}_P^F(V)$.

Next assume that $(ac, M') \in \mathrm{filter}_P^F(V)$, which amounts to $(\mu f.\lambda x.e, me) \in \mathit{Reach}_P^F$ and the existence of M such that $(ac, M) \in V$ and such that

$$M' = \left\{m \in M \mid (e, me[f, x \mapsto m]) \in \mathit{Reach}_P^F\right\}.$$

Then $ac \in dom(\mathcal{T}_F(V))$ and clearly $M' = dom(\mathcal{T}_F(V).ac)$, and (by Theorem 6.6) D_T contains a judgement for $\mu f.\lambda x.e$ at address me derived by $[\mathsf{fun}]^{w^\lambda}$ with w^λ of the form $(ac : \mathcal{T}_F(\{(ac, Mem_F)\}).ac)$ where by Lemma 6.2 we have $\mathcal{T}_F(\{(ac, Mem_F)\}) \leq_u \mathcal{T}_F(V)$ and hence $\mathcal{T}_F(\{(ac, Mem_F)\}).ac \leq_t \mathcal{T}_F(V).ac$. This demonstrates $(ac, M') \in \mathcal{F}_T(\mathcal{T}_F(V))$. □

Now observe that $\mathcal{C}_{F'}(l, me) = V\ (\neq \perp)$ iff D_T contains a judgement $A \vdash ue^l : u$ with address me and with $V = \mathcal{F}_T(u)$ iff $(ue^l, me) \in Reach_P^F$ with $u = \mathcal{T}_F(\mathcal{C}_F(l, me))$ and with $V = \mathcal{F}_T(u)$. So by Lemma E.1 we infer as desired that $\mathcal{C}_{F'}(l, me) = V$ iff $(ue^l, me) \in Reach_P^F$ and $V = \mathcal{F}_T(\mathcal{T}_F(\mathcal{C}_F(l, me))) = \mathrm{filter}_P^F(\mathcal{C}_F(l, me))$.

Similarly observe that $\rho_{F'}(z, m) = V\ (\neq \perp)$ iff there exists u with $\mathcal{T}_F^\rho(z, m) = A_T(z, m) = u$ and with $V = \mathcal{F}_T(u)$ iff $(z, m) \in Reach_P^F$ with $u = \mathcal{T}_F(\rho_F(z, m))$ and with $V = \mathcal{F}_T(u)$. So by Lemma E.1 we infer as desired that $\rho_{F'}(z, m) = V$ iff $(z, m) \in Reach_P^F$ and $V = \mathcal{F}_T(\mathcal{T}_F(\rho_F(z, m))) = \mathrm{filter}_P^F(\rho_F(z, m))$.

Now we consider the claim concerning $\Phi_{F'}$, where $ac = (\mu f.\lambda x.e_0, me_0)$ and l_2 is such that $e = ue_1^{l_1} @_l ue_2^{l_2}$ in $SubExpr_P$. If $\Phi_{F'}((l_2, me), ac) = K$ then there exists q such that D_T contains *(i)* a judgement for $\mu f.\lambda x.e_0$ with address me_0 derived by $[\mathsf{fun}]^{w^\lambda}$ with w^λ of the form $(q : t)$; *(ii)* a judgement for e with address me derived by $[\mathsf{app}]^{w^@}$ where $w^@(q) = K$. From the former judgement, we infer by Theorem 6.6 that $q = ac$. From the latter judgement, whose leftmost premise takes the form $A \vdash ue_1^{l_1} : \mathcal{T}_F(\mathcal{C}_F(l_1, me))$, we infer that $ac \in dom(\mathcal{T}_F(\mathcal{C}_F(l_1, me)))$ and by Theorem 6.6 that $w^@(ac) = \Phi_F((l_2, me), ac)$. This shows that $\Phi_F((l_2, me), ac) = K$ and that there exists M such that $(ac, M) \in \mathcal{C}_F(l_1, me)$. Clearly we also have that $(\mu f.\lambda x.e_0, me_0) \in Reach_P^F$ and $(e, me) \in Reach_P^F$.

Conversely, assume that $(\mu f.\lambda x.e_0, me_0) \in Reach_P^F$ and $(e, me) \in Reach_P^F$ and $(ac, M) \in \mathcal{C}_F(l_1, me)$ and $\Phi_F((l_2, me), ac) = K$. By Theorem 6.6 we infer that D_T contains at address me_0 a judgement for $\mu f.\lambda x.e_0$ derived by $[\mathsf{fun}]^{w^\lambda}$ with w^λ of the form $(ac : t)$; and (since $ac \in dom(\mathcal{T}_F(\mathcal{C}_F(l_1, me)))$) we also infer that D_T contains at address me a judgement for e with leftmost premise of the form $A \vdash ue_1^{l_1} : \mathcal{T}_F(\mathcal{C}_F(l_1, me))$ derived by $[\mathsf{app}]^{w^@}$ with $w^@(ac) = K$. This demonstrates that $\Phi_{F'}((l_2, me), ac) = K$. □

Proof of Corollary 7.3. In the following, we shall apply Theorem 7.2 to F as wells as to F'.

First we observe that $Mem_{F''} = Mem_{F'}$, that $Reach_P^{F'} = Reach_P^F$ and therefore $\mathrm{filter}_P^{F'} = \mathrm{filter}_P^F$, and that filter_P^F is idempotent.

Therefore $\mathcal{C}_{F''}(l, me) \neq \perp$ iff $\mathcal{C}_{F'}(l, me) \neq \perp$ and $(ue^l, me) \in Reach_P^{F'}$ iff $\mathcal{C}_F(l, me) \neq \perp$ and $(ue^l, me) \in Reach_P^F$ and $(ue^l, me) \in Reach_P^{F'}$ iff $\mathcal{C}_F(l, me) \neq \perp$ and $(ue^l, me) \in Reach_P^F$ iff $\mathcal{C}_{F'}(l, me) \neq \perp$, in which case we have $\mathcal{C}_{F''}(l, me) = \mathrm{filter}_P^{F'}(\mathcal{C}_{F'}(l, me)) = \mathrm{filter}_P^{F'}(\mathrm{filter}_P^F(\mathcal{C}_F(l, me))) = \mathrm{filter}_P^F(\mathrm{filter}_P^F(\mathcal{C}_F(l, me))) = \mathrm{filter}_P^F(\mathcal{C}_F(l, me)) = \mathcal{C}_{F'}(l, me)$. This shows $\mathcal{C}_{F''} = \mathcal{C}_{F'}$; in a similar way we see that $\rho_{F''} = \rho_{F'}$.

Finally, we must prove $\Phi_{F''} = \Phi_{F'}$ and it suffices to prove that whenever $\Phi_{F'}$ is defined then also $\Phi_{F''}$ is defined. So assume that $\Phi_{F'}((l_2, me), ac)$ is defined, where $ac = (\mu f.\lambda x.e_0, me_0)$ and l_2 is such that $e = ue_1^{l_1} @_l ue_2^{l_2}$ in $SubExpr_P$; then $(e, me) \in Reach_P^F$ and $(\mu f.\lambda x.e_0, me_0) \in Reach_P^F$ and for some M $(ac, M) \in \mathcal{C}_F(l_1, me)$. Still employing Theorem 7.2, we infer that $\mathcal{C}_{F'}(l_1, me) \neq \perp$ and that $\mathcal{C}_{F'}(l_1, me) = \mathrm{filter}_P^F(\mathcal{C}_F(l_1, me))$, so clearly there exists M' such that $(ac, M') \in \mathcal{C}_{F'}(l_1, me)$. Since $(e, me) \in Reach_P^{F'}$ and $(\mu f.\lambda x.e_0, me_0) \in Reach_P^{F'}$, this shows (by applying Theorem 7.2 on F') that $\Phi_{F''}((l_2, me), ac)$ is defined.

□

Proof of Theorem 7.6. Let $F = \mathcal{F}(T)$, so that $T' = \mathcal{T}(F)$. First note that F is valid and safe (by Theorem 5.6) so T' is uniform (by Theorem 6.6). Clearly $IT_{T'} = Mem_F = IT_T$.

Further observe (with $e = ue^l$) that $D_{T'}$ contains a judgement for e with address me iff $(e, me) \in Reach_P^F$ iff (by Theorem 5.6) $\mathcal{C}_F(l, me) \neq \perp$ iff D_T contains a judgement for e with address me.

Next we prove that $D_{T'}$ is strongly consistent. So let u occur in $D_{T'}$ with $q \neq q_{\text{int}}$ in $dom(u)$. Examining Definition 6.4 shows that u takes the form $\mathcal{T}_F(V)$ for some V, where V is in the range of either $\mathcal{C}_F$ or ρ_F; in both cases there exists u_1 such that $V = \mathcal{F}_T(u_1)$. From $u = \mathcal{T}_F(V)$ we infer that there exists (ac, M) in $\mathit{FlowVal}_F$ such that $q = ac$ and $(ac, M) \in V$; let $ac = (\mu f.\lambda x.e, me)$. From $(ac, M) \in V = \mathcal{F}_T(u_1)$ we infer that D_T contains a judgement for $\mu f.\lambda x.e$ with address me, and we have already seen that then also $D_{T'}$ contains a judgement for $\mu f.\lambda x.e$ with address me. By Theorem 6.6, this judgement is derived by $[\mathsf{fun}]^{w^\lambda}$ where $w^\lambda = (ac : \mathcal{T}_F(\{(ac, \mathit{Mem}_F)\}).ac)$. It is immediate from the definition of $\mathcal{T}_F$ that

$$\mathcal{T}_F(\{(ac, \mathit{Mem}_F)\}).ac \leq^c_\wedge \mathcal{T}_F(V).ac = \mathcal{T}_F(\{(ac, M)\}).ac$$

which shows the "at least one" part of Definition 7.5; the "at most one" part is obvious.

Finally, we assume that D_T is strongly consistent and must prove that $D_{T'}$ equals D_T—modulo renaming, so wlog. we can assume that if D_T contains a judgement for $\mu f.\lambda x.e_0$ with address me_0 derived by $[\mathsf{fun}]^{(q:t)}$ then $q = (\mu f.\lambda x.e_0, me_0)$. Our goal will be to show that

$$\text{if } u \text{ occurs in } D_T \text{ then } \mathcal{T}_F(\mathcal{F}_T(u)) = u \tag{1}$$

for then we can reason as follows: let D_T contain a judgement $A \vdash ue^l : u$ with address me; then the judgement for ue^l in $D_{T'}$ with address me will be of the form $A' \vdash ue^l : u'$ where

$$u' = \mathcal{T}_F(\mathcal{C}_F(l, me)) = \mathcal{T}_F(\mathcal{F}_T(u)) = u$$

and where $A' = \mathcal{T}^A_F(me)$. To see that A' equals A, note that

$$\mathcal{T}^A_F(\epsilon) = \epsilon \text{ and } \mathcal{T}^A_F(me[z \mapsto m]) = \mathcal{T}^A_F(me)[z \mapsto \mathcal{T}^\rho_F(z, m)] = \mathcal{T}^A_F(me)[z \mapsto \mathcal{T}_F(\mathcal{F}_T(A_T(z, m)))] = \mathcal{T}^A_F(me)[z \mapsto A_T(z, m)]$$

and that A equals $\mathsf{A}(me)$ where

$$\mathsf{A}(\epsilon) = \epsilon \text{ and } \mathsf{A}(me[z \mapsto m]) = \mathsf{A}(me)[z \mapsto A_T(z, m)].$$

So we now embark on proving (1); since $\mathcal{T}_F(\mathcal{F}_T(u_{\text{int}})) = u_{\text{int}}$ it is easy to see that this can be done by establishing that if u occurs in D_T and $q \neq q_{\text{int}} \in dom(u)$ then with $t = u.q$ the following holds:

$$\begin{array}{l}\text{if } t = \bigwedge_{k\in K}\{\{k\} : u_k \to u'_k\} \\ \text{then } \mathcal{T}_F(\mathcal{F}_T(\bigvee(q : t))) = \bigvee(q : \bigwedge_{k\in K}\{\{k\} : \mathcal{T}_F(\mathcal{F}_T(u_k)) \to \mathcal{T}_F(\mathcal{F}_T(u'_k))\}).\end{array} \tag{2}$$

So consider such q and t; our assumptions guarantee that D_T contains a judgement for $\mu f.\lambda x.e_0$ with address me_0 derived by $[\mathsf{fun}]^{(q:t_0)}$, where $q = (\mu f.\lambda x.e_0, me_0)$ and where $t_0 \leq^c_\wedge t$. The premises of this judgement are thus of the form

$$k \in K_0\text{: } A[f \mapsto u''_k][x \mapsto u_k] \vdash e_0 : u'_k \tag{3}$$

where $K \subseteq K_0$. By the "at most one" part of Definition 7.5 we infer that $\mathcal{F}_T(\bigvee(q : t)) = \{(q, K)\}$, so with $e_0 = ue_0^{l_0}$ we have

$$\mathcal{T}_F(\mathcal{F}_T(\bigvee(q : t))) = \bigvee(q : \bigwedge_{k\in K'}\{\{k\} : \mathcal{T}_F(\rho_F(x, k)) \to \mathcal{T}_F(\mathcal{C}_F(l_0, me_0[f, x \mapsto k]))\})$$

where K' is given by

$$\left\{k \in K \mid \mathit{Analyzes}^F_k(q)\right\}.$$

The judgements in (3) demonstrate (using Lemma C.1) that $K' = K$ and that for each $k \in K$ it holds that $\rho_F(x, k) = \mathcal{F}_T(u_k)$ and that $\mathcal{C}_F(l_0, me_0[f, x \mapsto k]) = \mathcal{F}_T(u'_k)$, thus establishing (2). □