

*Faithful Translations between Polyvariant Flows and Polymorphic Types**

TORBEN AMTOFT[†]

Boston University

FRANKLYN TURBAK[‡]

Wellesley College

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 (Palsberg & O’Keefe, 1995) demonstrated an equivalence between determining flow safety in the monovariant 0-CFA flow analysis and typability in a system with recursive types and subtyping (Amadio & Cardelli, 1993). Heintze showed equivalences between four restrictions of 0-CFA and four type systems parameterized by (1) subtyping and (2) recursive types (Heintze, 1995).

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 (Shivers,

* **DRAFT** as of March 2000.

[†] (*e-mail*: tamtsoft@bu.edu), Boston MA 02215, USA. Supported by NSF grant EIA-9806747.

[‡] (*e-mail*: fturbak@wellesley.edu), Wellesley MA 02481, USA. Supported by NSF grant EIA-9806747.

1991; Jagannathan & Weeks, 1995; Nielson & Nielson, 1997), polymorphic splitting (Wright & Jagannathan, 1998), type-directed flow analysis (Jagannathan *et al.*, 1997), and argument based polyvariance, such as Schmidt’s analysis (Schmidt, 1995) and Agesen’s cartesian product analysis (Agesen, 1995). In terms of the flow/type correspondence, several forms of flow polyvariance appear to correspond to type polymorphism expressed with intersection and union types (Banerjee, 1997; Wells *et al.*, 1997; Dimock *et al.*, 1997; Palsberg & Pavlopoulou, 1999). 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 (Palsberg & Pavlopoulou, 1999).

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 $e = (\lambda^1 x. \text{succ } x) @ ((\lambda^2 y. y) @ 3)$, 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 $v_1 = (\lambda x. \text{succ } x, [])$ and the only one reaching point 2 is $v_2 = (\lambda y. y, [])$. The flow-to-type translation of P&P yields the expected type derivation:

$$\frac{\dots \quad \frac{[] \vdash \lambda^1 x. \text{succ } x : \text{int} \rightarrow \text{int} \quad \frac{[] \vdash \lambda^2 y. y : \text{int} \rightarrow \text{int} \quad \dots}{[] \vdash (\lambda^2 y. y) @ 3 : \text{int}}}{[] \vdash (\lambda^1 x. \text{succ } x) @ ((\lambda^2 y. y) @ 3) : \text{int}}}{[] \vdash (\lambda^1 x. \text{succ } x) @ ((\lambda^2 y. y) @ 3) : \text{int}}$$

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 $\text{int} \rightarrow \text{int}$ 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, $\lambda^1 x. \text{succ } x$ has type $\text{int} \xrightarrow{\{1\}} \text{int}$ while $\lambda^2 y. y$ has type $\text{int} \xrightarrow{\{2\}} \text{int}$.

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 intersec-

tion 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) (Nielson & Nielson, 1997). 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((?; ?)), dead code elimination ((?)), defunctionalization((?; ?)), inlining((Wright & Jagannathan, 1998)), loop detection, partial evaluation((?)), object specialization ((?; ?)), run-time check elimination ((Wright & Jagannathan, 1998)), and uncurrying ((?)). 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.,(?; ?; ?)). 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 (Dimock *et al.*, 1997). 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 Project¹ compiler (Wells *et al.*, 1997).

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

2 The Language

We consider a language whose core is λ -calculus with recursion:

$$\begin{aligned} ue \in \mathbf{UnLabExpr} &::= z \mid \mu f. \lambda x. e \mid e @_l e \mid c \mid \mathbf{succ} \ e \mid \mathbf{if0} \ e \ \mathbf{then} \ e \ \mathbf{else} \ e \mid \dots \\ e \in \mathbf{LabExpr} &::= ue^l \\ l \in \mathbf{Lab} & \\ z \in \mathbf{Var} &::= x \mid f \\ x \in \mathbf{NVar} & \\ f \in \mathbf{RVar} & \end{aligned}$$

$\mu f. \lambda x. e$ denotes a function with parameter x which may call itself via f ; $\lambda x. e$ is a shorthand for $\mu f. \lambda x. e$ where f does not occur in e . Recursive variables (ranged over by f) and non-recursive variables (ranged over by x) are distinct; z ranges over both. There are also integer constants c , the successor function, and the ability to test for zero. Other constructs might be added, e.g., \mathbf{let}^2 .

All subexpressions have integer labels. We often write labels on constructors (e.g., write $\lambda^l x. e$ for $(\lambda x. e)^l$ and $e_1 @_l e_2$ for $(e_1 @ e_2)^l$). In examples, we use \mathbf{x} , \mathbf{y} and \mathbf{z} for non-recursive variables; \mathbf{g} is a non-recursive variable assumed to be bound to a function. We use E to range over closed expressions.

Example 1

The expression $P_1 \equiv (\lambda^6 \mathbf{g}. ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5)) @_0 (\lambda^8 \mathbf{x}. \mathbf{x}^7)$ shows the need for polyvariance: $\lambda^8 \mathbf{x}. \mathbf{x}^7$ is applied both to itself and to an integer.

Example 2 (PEP, pp. 12–14)

The following expression P_2 requires even more powerful polyvariance (assuming e_c is some unspecified expression):

$$(\lambda^6 \mathbf{g}. \mathbf{succ}^{13} ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5)) @_0 (\mathbf{if0}^9 \ e_c \ \mathbf{then} \ (\lambda^8 \mathbf{x}. \mathbf{x}^7) \ \mathbf{else} \ (\lambda^{12} \mathbf{y}. \lambda^{11} \mathbf{z}. \mathbf{z}^{10}))$$

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.

² Let-polymorphism can be simulated by intersection types.

An unlabeled function abstraction is conveniently represented as a *function* fn of the form $\mu f.\lambda x.e$. In examples, if f does not occur in e we shall omit f and just write $\lambda x.e$.

Like N&N, but unlike P&P, we use an environment-based small step semantics. This involves extending the syntax of expressions to incorporate “intermediate configurations”:

$$ue \in \mathbf{UnlabExpr} ::= \dots \mid \mathbf{bind} \ se \ \mathbf{in} \ e \mid \mathbf{close} \ fn \ \mathbf{in} \ se$$

This (re)definition is mutually recursive with the definitions below:

- a semantic environment se is an environment³ associating variables with semantic values.
- a semantic value sv is an *unlabeled* expression which is a constant c or of the form $\mathbf{close} \ fn \ \mathbf{in} \ se$.

We define $FV(\mathbf{bind} \ se \ \mathbf{in} \ e) = FV(\mathbf{close} \ fn \ \mathbf{in} \ se) = \emptyset$. This definition makes sense, since we impose the restriction that for $\mathbf{bind} \ se \ \mathbf{in} \ e$ it must hold that $FV(e) \subseteq \mathit{dom}(se)$ and for $\mathbf{close} \ fn \ \mathbf{in} \ se$ it must hold that $FV(fn) \subseteq \mathit{dom}(se)$. Note that for all semantic values sv we have $FV(sv) = \emptyset$.

Each step of the semantics is expressed as a judgement of the form $se \vdash e \Rightarrow e'$, which says that in environment se , e rewrites to e' in one step. The details, to be found in Appendix A, are basically as in N&N.

An expression not containing \mathbf{bind} or \mathbf{close} is said to be *pure*. For a function $\mu f.\lambda x.e$ we demand that e is pure. A program P is a pure, closed expression that is uniquely labeled, i.e., each label occurs at most once within P . A pleasant consequence of the latter property is that each subexpression of P (the set of which is denoted $SubExpr_P$) denotes a unique “position” within P .

Even though our main interest (the translations presented in Sections 5 and 6) lies in type/flow analyses for programs, we also have to consider how to analyze expressions that are not uniquely labeled or not pure. For if $[\] \vdash P \Rightarrow^* E$ it will in general not hold that E is a program. But it will hold that $E \in Exprs_P$, that is $Labs_E \subseteq Labs_P$, $Vars_E \subseteq Vars_P$, and $Funs_E \subseteq Funs_P$. Here $Labs_E$ denotes the set of labels occurring in E , $Vars_E$ denotes the set of variables (free or bound) in E , and $Funs_E$ denotes the set of functions induced by E . That is, $\mu f.\lambda x.e$ belongs to $Funs_E$ iff E contains an expression of the form $\mu f.\lambda x.e$ or of the form $\mathbf{close} \ \mu f.\lambda x.e$ in se .

Example 3

With $ue_1 = \mu f.\lambda x.e_1$ and $P = \mathbf{if}0^l \ \mathbf{then} \ ue_1^{l1} \ \mathbf{else} \ 7$, we have $[\] \vdash P \Rightarrow ue_1^l$. And in fact $ue_1^l \in Exprs_P$ holds (since the outermost label of a function abstraction is not included in a function).

³ Environments (of which we shall use several kinds) are represented as a list of entries, each of the form $[z \mapsto y]$ where y ranges over the codomain of the environment. If $(z \mapsto y) \in \mathcal{E}$ we say that $z \in \mathit{dom}(\mathcal{E})$, and if $[z \mapsto y]$ is the rightmost entry for z in \mathcal{E} we write $y = \mathcal{E}(z)$. Note that $\mathcal{E}[z \mapsto y](z')$ equals y if $z = z'$ and equals $\mathcal{E}(z')$ otherwise. We write $\mathcal{E}[z_1, z_2 \mapsto y]$ for $\mathcal{E}[z_1 \mapsto y, z_2 \mapsto y]$.

3 The Type System

Types are built from base types, function types, intersection types, and union types as follows (where **ITag** and **UTag** are unspecified):

$$\begin{aligned}
 t \in \mathbf{ElementaryType} &::= \mathbf{int} \mid \bigwedge_{i \in I} \{K_i : u_i \rightarrow u'_i\} \\
 u \in \mathbf{UnionType} &::= \bigvee_{i \in I} \{q_i : t_i\} \\
 K &\in \mathcal{P}(\mathbf{ITag}) \\
 k &\in \mathbf{ITag} \\
 q &\in \mathbf{UTag}
 \end{aligned}$$

Such grammars are usually interpreted inductively, but this one is to be viewed co-inductively. That is, types are regular (possibly infinite) trees formed according to the above specification. Two types are considered equal if their infinite unwindings are equal (modulo renaming of the index sets I)⁴.

An elementary type t is either an integer **int** or an intersection type of the form $\bigwedge_{i \in I} \{K_i : u_i \rightarrow u'_i\}$, where I is a (possibly empty) finite index set, where each u_i and u'_i is a union type, and where the K_i 's, known as *I-tagsets*, are non-empty finite disjoint sets of *I-tags*. We write $\mathit{dom}(t)$ for $\bigcup_{i \in I} K_i$. Intuitively, if an expression e has the above intersection type then *for all* $i \in I$ it holds that the expression maps values of type u_i into values of type u'_i . This is the sense in which intersection types are considered finite 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, where each t_i is an elementary type, and where the q_i are distinct *U-tags*. We write $\mathit{dom}(u)$ for $\bigcup_{i \in I} \{q_i\}$, 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 $t_i = \mathbf{int}$ iff $q_i = q_{\mathbf{int}}$ where $q_{\mathbf{int}}$ 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 above union type then *there exists* an $i \in I$ such that e has the elementary type t_i . This is the sense in which union types are considered finite existential types.

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

The types are much as in P&P except for the presence of tags. These annotations serve as witnesses for existentials 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 help to avoid the precision-losing merging seen in P&P’s type-to-flow translation (cf. Sect. 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 our intersection and union types (unlike those of P&P) are not associative, commutative, or idempotent (ACI) due to U-tags and I-tagsets, in the

⁴ An example: if $q_1 = q'_4$ 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 $\bigvee_{i \in \{3,4\}} \{q'_i : t'_i\}$.

sense that while $\bigvee(q_1 : t_1, q_2 : t_2)$ equals $\bigvee(q_2 : t_2, q_1 : t_1)$ it does not equal $\bigvee(q_1 : t_2, q_2 : t_1)$.

3.1 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 $u_1 \rightarrow u'_1$ and $u_2 \rightarrow u'_2$ also

- can be assigned one of them, i.e., for $i \in \{1, 2\}$, $\bigwedge(K_1 : u_1 \rightarrow u'_1, K_2 : u_2 \rightarrow u'_2) \leq_t \bigwedge(K_i : u_i \rightarrow u'_i)$;
- can be assigned a function type that “covers” both, i.e., $\bigwedge(K_1 : u_1 \rightarrow u'_1, K_2 : u_2 \rightarrow u'_2) \leq_t \bigwedge(K_1 \cup K_2 : u_{12} \rightarrow u'_{12})$ 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:

$$\begin{aligned} & \bigvee_{i \in I} \{q_i : t_i\} \leq_u \bigvee_{j \in J} \{q'_j : t'_j\} \\ & \quad \text{iff for all } i \in I \text{ there exists } j \in J \text{ such that } q_i = q'_j \text{ and } t_i \leq_t t'_j \\ & \mathbf{int} \leq_t \mathbf{int} \\ & \bigwedge_{i \in I} \{K_i : u_i \rightarrow u'_i\} \leq_t \bigwedge_{j \in J} \{K'_j : u''_j \rightarrow u'''_j\} \\ & \quad \text{iff for all } j \in J \text{ there exists } I_0 \subseteq I \text{ such that} \\ & \quad \quad K'_j = \bigcup_{i \in I_0} K_i \text{ and } \forall i \in I_0. u'_i \leq_u u'''_j \text{ and} \\ & \quad \quad \forall q \in \text{dom}(u''_j). \exists i \in I_0. q \in \text{dom}(u_i) \text{ and } u''_j.q \leq_t u_i.q. \end{aligned}$$

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 lattice⁵. 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 B) yields:

Lemma 3.1

The relations \leq_u and \leq_t are reflexive and transitive.

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 $u_1 = \bigvee(0 : \bigwedge(\{1\} : u_{\text{int}} \rightarrow u_1))$ and $u_2 = \bigvee(0 : \bigwedge(\{1\} : u_{\text{int}} \rightarrow u_2, \{2\} : u_{\text{int}} \rightarrow u_{\text{int}}))$.

Observe that if $t \leq_t t'$, then $\text{dom}(t') \subseteq \text{dom}(t)$, and that if $t = \bigwedge_{i \in I} \{K_i : u_i \rightarrow u'_i\}$ and $t' = \bigwedge_{i \in I'} \{K_i : u_i \rightarrow u'_i\}$ 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 for search during type checking. Moreover, our ordering seems more natural than 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.

3.2 Typing Rules

A *typing* T for a program P is a tuple (P, IT_T, UT_T, D_T) , where IT_T is a finite set of I-tags, UT_T is a finite set of U-tags, and D_T is a derivation of $[] \vdash P : u$ according to the inference rules given in Fig. 1 and briefly explained below. In a judgement $A \vdash e : u$, A is an environment with bindings of the form $[z \mapsto u]$; 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 . The rules for intermediate configurations employ a predicate \bowtie that is defined as follows:

$$se \bowtie A \text{ iff } \forall z \in \text{dom}(se). [] \vdash se(z) : A(z).$$

Note that all rules are “structural”, in particular subtyping has been “inlined” in all of them so as to simplify the type/flow correspondence, and that the typing of an expression ue^l does not depend on l which may therefore be omitted.

The rules for function abstraction and function application are both instrumented with a “witness” that enables reconstructing the justification for applying the rule. In $[\text{app}]^{w^\circ}$, the type of the operator is a (possibly empty) union, all components of which have the expected function type but the I-tagsets may differ; the *app-witness* w° is a partial mapping from $\text{dom}(u_1)$ that given q produces the corresponding I-tagset. In $[\text{fun}]^{w^\lambda}$, the function types resulting from analyzing the body in several different environments 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 this inference is the pair $(q : t)$. Note that K may be empty in which case the body is not analyzed (letting flows “have it their way”).

Example 4

For the program P_1 from Ex. 1, we can construct a typing T_1 as follows: $IT_{T_1} = \{0, 1, 2\}$, $UT_{T_1} = \{q_x, q_g\}$, and D_{T_1} is as in Fig. 2, where

$$\begin{aligned} u'_x &= \vee(q_x : \bigwedge(\{1\} : u_{\text{int}} \rightarrow u_{\text{int}})) \\ u_x &= \vee(q_x : \bigwedge(\{1\} : u_{\text{int}} \rightarrow u_{\text{int}}, \{2\} : u'_x \rightarrow u'_x)) \\ u_g &= \vee(q_g : \bigwedge(\{0\} : u_x \rightarrow u_{\text{int}})) \\ A_g &= [g \mapsto u_x] \quad A_x = [x \mapsto u_{\text{int}}] \quad A'_x = [x \mapsto u'_x] \end{aligned}$$

[var]	$A \vdash z^l : u$	if $A(z) \leq_u u$
[fun] ^(q:t)	$\frac{\forall k \in K : A[f \mapsto u''_k, x \mapsto u_k] \vdash e : u'_k}{A \vdash \mu f. \lambda^l x. e : u}$	if $t = \bigwedge_{k \in K} \{\{k\} : u_k \rightarrow u'_k\}$ $\wedge \bigvee (q : t) \leq_u u$ $\wedge \forall k \in K. \bigvee (q : t) \leq_u u'_k$
[app] ^{w[@]}	$\frac{A \vdash e_1 : u_1 \quad A \vdash e_2 : u_2}{A \vdash e_1 @_l e_2 : u}$	if $\forall q \in \text{dom}(u_1). u_1.q \leq_t \wedge (w^{\text{@}}(q) : u_2 \rightarrow u)$
[con]	$A \vdash c^l : u$	if $u_{\text{int}} \leq_u u$
[suc]	$\frac{A \vdash e_1 : u_1}{A \vdash \text{succ}^l e_1 : u}$	if $u_1 \leq_u u_{\text{int}} \leq_u u$
[if]	$\frac{A \vdash e_0 : u_0 \quad A \vdash e_1 : u_1 \quad A \vdash e_2 : u_2}{A \vdash \text{if}0^l e_0 \text{ then } e_1 \text{ else } e_2 : u}$	if $u_0 \leq_u u_{\text{int}} \wedge u_1 \leq_u u \wedge u_2 \leq_u u$
[bind]	$\frac{se \bowtie A' \quad A' \vdash e : u'}{A \vdash \text{bind}^l se \text{ in } e : u}$	if $u' \leq_u u$
[clos]	$\frac{se \bowtie A' \quad A' \vdash fn : u}{A \vdash \text{close}^l fn \text{ in } se : u}$	

Fig. 1. The typing rules

$$\frac{\frac{\frac{A_g \vdash \mathbf{g}^3 : u_x \quad A_g \vdash \mathbf{g}^4 : u'_x}{A_g \vdash \mathbf{g}^3 @_2 \mathbf{g}^4 : u'_x} \quad A_g \vdash 0^5 : u_{\text{int}}}{A_g \vdash (\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5 : u_{\text{int}}}}{\frac{[] \vdash \lambda^6 \mathbf{g}. ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5) : u_g \quad \frac{A_x \vdash \mathbf{x}^7 : u_{\text{int}} \quad A'_x \vdash \mathbf{x}^7 : u'_x}{[] \vdash \lambda^8 \mathbf{x}. \mathbf{x}^7 : u_x}}{[] \vdash (\lambda^6 \mathbf{g}. ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5)) @_0 (\lambda^8 \mathbf{x}. \mathbf{x}^7) : u_{\text{int}}}}$$

 Fig. 2. A derivation D_{T_1} for the program P_1 from Example 1.

Note that $u_x \leq_u u'_x$, and that $u_x.q_x \leq_t \wedge (\{2\} : u'_x \rightarrow u'_x)$ so that $\{q_x \mapsto \{2\}\}$ is indeed an **app**-witness for the inference at the top left of Fig. 2.

Example 5

For the program P_2 from Ex. 2, we can construct a typing T_2 where $IT_{T_2} = \{1, 2, 3, \mathbf{x}, \mathbf{y}\}$ and $UT_{T_2} = \{q_x, q_y, q_z, q_g\}$. To see this, note that Fig. 3 demonstrates that we have the judgements

$$\begin{aligned} & [] \vdash \lambda^6 \mathbf{g}. \text{succ}^{13} ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5) : \bigvee (q_g : \bigwedge (\{\mathbf{x}\} : u_x \rightarrow u_{\text{int}}, \{\mathbf{y}\} : u_y \rightarrow u_{\text{int}})) \\ & \text{and} \\ & [] \vdash \text{if}0^9 e_c \text{ then } (\lambda^8 \mathbf{x}. \mathbf{x}^7) \text{ else } (\lambda^{12} \mathbf{y}. \lambda^{11} \mathbf{z}. \mathbf{z}^{10}) : u_{xy} \end{aligned}$$

which form a valid set of premises for $[\text{app}]^{\{q_g \mapsto \{\mathbf{x}, \mathbf{y}\}\}}$ since

$$\bigwedge (\{\mathbf{x}\} : u_x \rightarrow u_{\text{int}}, \{\mathbf{y}\} : u_y \rightarrow u_{\text{int}}) \leq_t \bigwedge (\{\mathbf{x}, \mathbf{y}\} : u_{xy} \rightarrow u_{\text{int}})$$

Here u'_x and u_x are as in Example 4, and additionally

$$u_z = \bigvee (q_z : \bigwedge (\{1\} : u_{\text{int}} \rightarrow u_{\text{int}}))$$

$$\begin{array}{c}
\frac{\dots}{[\mathbf{g} \mapsto u_x] \vdash (\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5 : u_{\text{int}}} \text{ (Ex.4)} \quad \frac{[\mathbf{g} \mapsto u_y] \vdash \mathbf{g}^3 : u_y \quad [\mathbf{g} \mapsto u_y] \vdash \mathbf{g}^4 : u'_y}{[\mathbf{g} \mapsto u_y] \vdash \mathbf{g}^3 @_2 \mathbf{g}^4 : u_z} \quad \dots}{[\mathbf{g} \mapsto u_y] \vdash (\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5 : u_{\text{int}}} \\
\frac{[\mathbf{g} \mapsto u_x] \vdash \text{succ}^{13} ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5) : u_{\text{int}}}{[\mathbf{g} \mapsto u_x] \vdash \text{succ}^{13} ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5) : u_{\text{int}}} \quad \frac{[\mathbf{g} \mapsto u_y] \vdash \text{succ}^{13} ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5) : u_{\text{int}}}{[\mathbf{g} \mapsto u_y] \vdash \text{succ}^{13} ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5) : u_{\text{int}}} \\
\hline
[] \vdash \lambda^6 \mathbf{g} . \text{succ}^{13} ((\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5) : \bigvee (q_{\mathbf{g}} : \bigwedge (\{\mathbf{x}\} : u_x \rightarrow u_{\text{int}}, \{\mathbf{y}\} : u_y \rightarrow u_{\text{int}}))
\end{array}$$

$$\begin{array}{c}
\frac{\dots}{[] \vdash \lambda^8 \mathbf{x} . \mathbf{x}^7 : u_x} \text{ (Ex.4)} \quad \frac{[\mathbf{y} \mapsto u'_y] \vdash \lambda^{11} \mathbf{z} . \mathbf{z}^{10} : u_z}{[] \vdash \lambda^{12} \mathbf{y} . \lambda^{11} \mathbf{z} . \mathbf{z}^{10} : u_y} \\
\hline
[] \vdash \text{if}^9 e_c \text{ then } (\lambda^8 \mathbf{x} . \mathbf{x}^7) \text{ else } (\lambda^{12} \mathbf{y} . \lambda^{11} \mathbf{z} . \mathbf{z}^{10}) : u_{xy}
\end{array}$$

Fig. 3. Typing the program P_2 from Example 2.

$$\begin{aligned}
u'_y &= \bigvee (q_y : \bigwedge ()) \\
u_y &= \bigvee (q_y : \bigwedge (\{\mathbf{3}\} : u'_y \rightarrow u_z)) \\
u_{xy} &= \bigvee (q_x : \bigwedge (\{\mathbf{1}\} : u_{\text{int}} \rightarrow u_{\text{int}}, \{\mathbf{2}\} : u'_x \rightarrow u'_x), q_y : \bigwedge (\{\mathbf{3}\} : u'_y \rightarrow u_z))
\end{aligned}$$

Note that the types are all finite, that $u_y \leq_u u'_y$, and that for all $q \in \text{dom}(u_{xy})$ it holds that either $u_{xy}.q = u_x.q$ or $u_{xy}.q = u_y.q$.

3.3 Semantic soundness

The type system in Fig. 1 satisfies a subject reduction property, proved in Appendix B, which on top-level reads: If $[] \vdash E \Rightarrow E'$ and $[] \vdash E : u$ then also $[] \vdash E' : u$:

Theorem 3.2

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 $[] \vdash P : u$ and that $[] \vdash P \Rightarrow E$, where E is “stuck” in that E is not a semantic value and yet for no E' it holds that $[] \vdash E \Rightarrow E'$. By (repeated applications of) Theorem 3.2 we infer that $[] \vdash E : u$. A case analysis reveals that within an “evaluation context” of E there exists an expression e that is of the form either

$$c @_l sv \text{ or } \text{succ}^l (\text{close}^{l'} fn \text{ in } se) \text{ or } \text{if}^l (\text{close}^{l'} fn \text{ in } se) \text{ then } e_1 \text{ else } e_2.$$

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 impossible⁷, and as desired we have arrived at a contradiction.

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

3.4 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 6

In D_{T_1} (Fig. 2), the judgement $A_x \vdash x^7 : u_{\text{int}}$ has address $[x \mapsto 1]$ and the judgement $A'_x \vdash x^7 : u'_x$ has address $[x \mapsto 2]$.

To be more formal: the root of a derivation D_T has address $[\]$; 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 7

For T_1 we have, e.g., $A_{T_1}(x, 1) = u_{\text{int}}$ and $A_{T_1}(x, 2) = u'_x$.

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 $(P, \text{Mem}_F, \mathcal{C}_F, \rho_F, \Phi_F)$, 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 ($m \in \text{Mem}_F$) 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 ($me \in \text{MemEnv}_F$) take the form $[z \mapsto m]$ with m in Mem_F . Accordingly, a *flow configuration* ($\in \text{FlowConf}_F$) is a pair (e, me) ($e \in \text{Exps}_P$ and $me \in \text{MemEnv}_F$), where $FV(e) \subseteq \text{dom}(me)$.

The goal of the flow analysis is to associate a set of *flow values* to each configuration, where a flow value ($v \in \text{FlowVal}_F$) is either an integer Int or of the form (ac, M) , where ac ($\in \text{AbsClos}_F$) is an *abstract closure* of the form (fn, me) with $fn \in \text{Funs}_P$ and $FV(fn) \subseteq \text{dom}(me)$, and where $M \subseteq \text{Mem}_F$. The M component can be thought of 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 $M = Mem_F$ everywhere.

A *flow set* V ($\in FlowSet_F$) 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 $FlowSet_F$ 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 $FlowVal_F$ is defined by stipulating that $Int \leq_v Int$ 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 (Wells *et al.*, 1997). It is easy to see that \leq_V is reflexive and transitive, and that it makes $FlowSet_F$ a complete lattice. Note that $Int \in \prod_{i \in I}^V V_i$ iff for all $i \in I$ it holds that $Int \in V_i$; and that $(ac, M) \in \prod_{i \in I}^V V_i$ 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 $v \in \prod_{i \in I}^V V_i$ 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 ($uv \in UnAnnFlowVal_F$) 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, $\epsilon_v(Int) = Int$ and $\epsilon_v((ac, M)) = ac$ and $\iota_v(Int) = Int$ and $\iota_v(ac) = (ac, Mem_F)$; 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 $FlowSet_F$ and $\mathcal{P}(UnAnnFlowVal_F)$. 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 $(Labs_P \times MemEnv_F) \times AbsClos_F$ to $\mathcal{P}(Mem_F)$. 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 $Labs_P \times MemEnv_F$ to $(FlowSet_F)_\perp$. 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 semantic 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 semantic 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 $FlowSet_F$ is lifted to a relation \leq_V on $FlowSet_{F_\perp}$; note that $\perp \leq_V \emptyset$ is true whereas $\emptyset \leq_V \perp$ is false. $FlowSet_{F_\perp}$ is a complete lattice, and $\prod_{i \in I}^V V_i = \perp$ iff there exists an $i \in I$ such that $V_i = \perp$. Also the predicate \in can be lifted⁸ in the natural way, so that, e.g., $v \notin \perp$ is considered a true statement.

Example 8

For the program P_1 from Ex. 1, a flow analysis F_1 with $Mem_{F_1} = \{0, 1, 2\}$ is given below. We have named some entities (note that $v_x \leq_v v'_x$):

⁸ 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.

$$\begin{array}{lll}
me_{\mathbf{g}} = [\mathbf{g} \mapsto 0] & ac_{\mathbf{g}} = (\lambda \mathbf{g} \cdot \dots, []) & v_{\mathbf{g}} = (ac_{\mathbf{g}}, \{0\}) \\
me_{x_1} = [\mathbf{x} \mapsto 1] & ac_{\mathbf{x}} = (\lambda \mathbf{x} \cdot \mathbf{x}^7, []) & v'_{\mathbf{x}} = (ac_{\mathbf{x}}, \{1\}) \\
me_{x_2} = [\mathbf{x} \mapsto 2] & & v_{\mathbf{x}} = (ac_{\mathbf{x}}, \{1, 2\})
\end{array}$$

\mathcal{C}_{F_1} and ρ_{F_1} are given by the entries below (all other are \perp):

$$\begin{array}{l}
\{v_{\mathbf{g}}\} = \mathcal{C}_{F_1}(6, []) \\
\{\text{Int}\} = \rho_{F_1}(\mathbf{x}, 1) = \mathcal{C}_{F_1}(7, me_{x_1}) = \mathcal{C}_{F_1}(5, me_{\mathbf{g}}) = \mathcal{C}_{F_1}(1, me_{\mathbf{g}}) = \mathcal{C}_{F_1}(0, []) \\
\{v'_{\mathbf{x}}\} = \rho_{F_1}(\mathbf{x}, 2) = \mathcal{C}_{F_1}(7, me_{x_2}) = \mathcal{C}_{F_1}(4, me_{\mathbf{g}}) = \mathcal{C}_{F_1}(2, me_{\mathbf{g}}) \\
\{v_{\mathbf{x}}\} = \rho_{F_1}(\mathbf{g}, 0) = \mathcal{C}_{F_1}(3, me_{\mathbf{g}}) = \mathcal{C}_{F_1}(8, [])
\end{array}$$

Thus $(\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5$ is analyzed with \mathbf{g} bound to 0, and \mathbf{x}^7 is analyzed twice: with \mathbf{x} bound to 1 and with \mathbf{x} bound to 2. Accordingly, Φ_{F_1} is given by

$$\Phi_{F_1}((8, []), ac_{\mathbf{g}}) = \{0\}, \Phi_{F_1}((5, me_{\mathbf{g}}), ac_{\mathbf{x}}) = \{1\}, \Phi_{F_1}((4, me_{\mathbf{g}}), ac_{\mathbf{x}}) = \{2\}.$$

Example 9

For the program P_2 from Ex. 2, a flow analysis F_2 with $Mem_{F_2} = \{1, 2, 3, \mathbf{x}, \mathbf{y}\}$ is given below. We have named some entities:

$$\begin{array}{lll}
me_{\mathbf{g}\mathbf{x}} = [\mathbf{g} \mapsto \mathbf{x}] & ac_{\mathbf{g}} = (\lambda \mathbf{g} \cdot \dots, []) & v_{\mathbf{g}} = (ac_{\mathbf{g}}, \{\mathbf{x}, \mathbf{y}\}) \\
me_{\mathbf{g}\mathbf{y}} = [\mathbf{g} \mapsto \mathbf{y}] & & v'_{\mathbf{x}} = (ac_{\mathbf{x}}, \{1\}) \\
me_{x_1} = [\mathbf{x} \mapsto 1] & ac_{\mathbf{x}} = (\lambda \mathbf{x} \cdot \mathbf{x}^7, []) & v_{\mathbf{x}} = (ac_{\mathbf{x}}, \{1, 2\}) \\
me_{x_2} = [\mathbf{x} \mapsto 2] & & v'_{\mathbf{y}} = (ac_{\mathbf{y}}, \emptyset) \\
me_{\mathbf{y}} = [\mathbf{y} \mapsto 3] & ac_{\mathbf{y}} = (\lambda \mathbf{y} \cdot \lambda^{11} \mathbf{z} \cdot \mathbf{z}^{10}, []) & v_{\mathbf{y}} = (ac_{\mathbf{y}}, \{3\}) \\
me_{\mathbf{z}} = [\mathbf{y} \mapsto 3, \mathbf{z} \mapsto 1] & ac_{\mathbf{z}} = (\lambda \mathbf{z} \cdot \mathbf{z}^{10}, me_{\mathbf{y}}) & v_{\mathbf{z}} = (ac_{\mathbf{z}}, \{1\})
\end{array}$$

\mathcal{C}_{F_2} and ρ_{F_2} are given by the entries below (all other are \perp):

$$\begin{array}{l}
\{v_{\mathbf{g}}\} = \mathcal{C}_{F_2}(6, []) \\
\{\text{Int}\} = \rho_{F_2}(\mathbf{x}, 1) = \rho_{F_2}(\mathbf{z}, 1) = \mathcal{C}_{F_2}(7, me_{x_1}) = \mathcal{C}_{F_2}(10, me_{\mathbf{z}}) \\
= \mathcal{C}_{F_2}(5, me_{\mathbf{g}\mathbf{x}}) = \mathcal{C}_{F_2}(5, me_{\mathbf{g}\mathbf{y}}) = \mathcal{C}_{F_2}(1, me_{\mathbf{g}\mathbf{x}}) = \mathcal{C}_{F_2}(1, me_{\mathbf{g}\mathbf{y}}) \\
= \mathcal{C}_{F_2}(13, me_{\mathbf{g}\mathbf{x}}) = \mathcal{C}_{F_2}(13, me_{\mathbf{g}\mathbf{y}}) = \mathcal{C}_{F_2}(0, []) \\
\{v'_{\mathbf{x}}\} = \rho_{F_2}(\mathbf{x}, 2) = \mathcal{C}_{F_2}(7, me_{x_2}) = \mathcal{C}_{F_2}(4, me_{\mathbf{g}\mathbf{x}}) = \mathcal{C}_{F_2}(2, me_{\mathbf{g}\mathbf{x}}) \\
\{v_{\mathbf{x}}\} = \rho_{F_2}(\mathbf{g}, \mathbf{x}) = \mathcal{C}_{F_2}(3, me_{\mathbf{g}\mathbf{x}}) = \mathcal{C}_{F_2}(8, []) \\
\{v'_{\mathbf{y}}\} = \rho_{F_2}(\mathbf{y}, 3) = \mathcal{C}_{F_2}(4, me_{\mathbf{g}\mathbf{y}}) \\
\{v_{\mathbf{y}}\} = \rho_{F_2}(\mathbf{g}, \mathbf{y}) = \mathcal{C}_{F_2}(3, me_{\mathbf{g}\mathbf{y}}) = \mathcal{C}_{F_2}(12, []) \\
\{v_{\mathbf{z}}\} = \mathcal{C}_{F_2}(2, me_{\mathbf{g}\mathbf{y}}) = \mathcal{C}_{F_2}(11, me_{\mathbf{y}}) \\
\{v_{\mathbf{x}}, v_{\mathbf{y}}\} = \mathcal{C}_{F_2}(9, [])
\end{array}$$

We see that $(\mathbf{g}^3 @_2 \mathbf{g}^4) @_1 0^5$ is analyzed twice: with \mathbf{g} bound to \mathbf{x} , and with \mathbf{g} bound to \mathbf{y} . And in fact, Φ_{F_2} is given by

$$\begin{array}{l}
\Phi_{F_2}((9, []), ac_{\mathbf{g}}) = \{\mathbf{x}, \mathbf{y}\} \\
\Phi_{F_2}((5, me_{\mathbf{g}\mathbf{x}}), ac_{\mathbf{x}}) = \{1\}, \Phi_{F_2}((5, me_{\mathbf{g}\mathbf{y}}), ac_{\mathbf{z}}) = \{1\} \\
\Phi_{F_2}((4, me_{\mathbf{g}\mathbf{x}}), ac_{\mathbf{x}}) = \{2\}, \Phi_{F_2}((4, me_{\mathbf{g}\mathbf{y}}), ac_{\mathbf{y}}) = \{3\}
\end{array}$$

$$\begin{array}{l}
\text{[var]} \quad F \models^{me} z^l \text{ iff } \perp \neq \rho_F(z, me(z)) \leq_V \mathcal{C}_F(l, me) \\
\text{[fun]} \quad F \models^{me} \mu f. \lambda x. e_0 \text{ iff } \{((\mu f. \lambda x. e_0), me), Mem_F\} \leq_V \mathcal{C}_F(l, me) \\
\text{[app]} \quad F \models^{me} ue_1^{l_1} @_l ue_2^{l_2} \text{ iff} \\
\quad \mathcal{C}_F(l, me) \neq \perp \wedge F \models^{me} ue_1^{l_1} \wedge F \models^{me} ue_2^{l_2} \wedge \\
\quad \forall (ac_0, M_0) \in \mathcal{C}_F(l_1, me) \\
\quad \quad \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} \\
\quad \quad M \subseteq M_0 \wedge \forall v \in \mathcal{C}_F(l_2, me). \exists m \in M. \{v\} \leq_V \rho_F(x, m) \wedge \\
\quad \quad \forall m \in M: F \models^{me_0[f, x \mapsto m]} ue_0^{l_0} \wedge \\
\quad \quad \mathcal{C}_F(l_0, me_0[f, x \mapsto m]) \leq_V \mathcal{C}_F(l, me) \wedge \\
\quad \quad \rho_F(x, m) \neq \perp \wedge \{(ac_0, Mem_F)\} \leq_V \rho_F(f, m) \\
\text{[con]} \quad F \models^{me} c^l \text{ iff } \text{Int} \in \mathcal{C}_F(l, me) \\
\text{[suc]} \quad F \models^{me} \text{succ}^l e_1 \text{ iff } F \models^{me} e_1 \wedge \text{Int} \in \mathcal{C}_F(l, me) \\
\text{[if]} \quad F \models^{me} \text{if} 0^l e_0 \text{ then } ue_1^{l_1} \text{ else } ue_2^{l_2} \text{ iff} \\
\quad F \models^{me} e_0 \wedge F \models^{me} ue_1^{l_1} \wedge F \models^{me} ue_2^{l_2} \wedge \\
\quad \mathcal{C}_F(l_1, me) \leq_V \mathcal{C}_F(l, me) \wedge \mathcal{C}_F(l_2, me) \leq_V \mathcal{C}_F(l, me) \\
\text{[bind]} \quad F \models^{me} \text{bind}^l se \text{ in } ue_1^{l_1} \text{ iff} \\
\quad \exists me_1 \text{ with } se \mathcal{R}_F me_1: \\
\quad \quad F \models^{me_1} ue_1^{l_1} \wedge \mathcal{C}_F(l_1, me_1) \leq_V \mathcal{C}_F(l, me) \\
\text{[clos]} \quad F \models^{me} \text{close}^l fn \text{ in } se \text{ iff} \\
\quad \exists me_0 \text{ with } se \mathcal{R}_F me_0: \{((fn, me_0), Mem_F)\} \leq_V \mathcal{C}_F(l, me)
\end{array}$$

Fig. 4. 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 $(e, me) \in FlowConf_F$. The predicate must satisfy the specification in Fig. 4, where the clause for intermediate configurations employs a predicate \mathcal{R}_F that is defined mutually recursively with another predicate \mathcal{V}_F :

$$\begin{array}{l}
se \mathcal{R}_F me \quad \text{iff} \quad \forall z \in dom(se): se(z) \mathcal{V}_F \rho_F(z, me(z)) \\
c \mathcal{V}_F V \quad \text{iff} \quad \text{Int} \in V
\end{array}$$

$$(\text{close } fn \text{ in } se) \mathcal{V}_F V \quad \text{iff} \quad \exists me \text{ with } se \mathcal{R}_F me: \{((fn, me), Mem_F)\} \leq_V V$$

The specification in Fig. 4 gives rise to a monotone functional \mathcal{G}_F on the complete lattice $\mathcal{P}(FlowConf_F)$; 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 $FV(\mu f. \lambda x. e_0)$. 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 (Agesen, 1995). In N&N's framework, M is always a singleton $\{m\}$; in that case the condition

“ $\forall v \in \mathcal{C}_F(l_2, me). \dots$ ” 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.

Note that even if e is a subexpression of a program P with $F \models^{\square} P$ 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 $\mathcal{C}_F(l, me) \neq \perp$. We would also like the converse implication to hold:

Definition 4.1

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

4.2 Semantic soundness.

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

Theorem 4.2

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 4.2 we infer $F \models^{me} \mu f.\lambda^l x.e_0$, that is $\{((\mu f.\lambda x.e_0, me), \text{Mem}_F)\} \leq_V \mathcal{C}_F(l, me)$. Note that this result hinges on the fact that $se \vdash ue^l \Rightarrow ue'^l$ implies $l = l'$ (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 4.3

Let a flow analysis F for P be given. We say that F is safe iff for all ue^l in SubExpr_P and for all me it holds: (i) if $ue = ue_1^{l_1} @ e_2$ then $\text{Int} \notin \mathcal{C}_F(l_1, me)$; (ii) if $ue = \text{succ } ue_1^{l_1}$ then $v \in \mathcal{C}_F(l_1, me)$ implies $v = \text{Int}$; (iii) if $ue = \text{if } 0 \text{ } ue_0^{l_0} \text{ then } e_1 \text{ else } e_2$ then $v \in \mathcal{C}_F(l_0, me)$ implies $v = \text{Int}$.

Example 10

Referring back to Examples 8 and 9, it clearly holds that F_1 is safe and F_2 is safe, and it is easy (though a little cumbersome) to verify that F_1 is valid and F_2 is valid.

4.3 Taxonomy of Flow Analyses

Two common categories of flow analyses are the “call-string based” (e.g., (Shivers, 1991)) and the “argument-based” (e.g., (Schmidt, 1995; Agesen, 1995)). Below we shall see that our *descriptive* framework can model both approaches (which can be “mixed”, as in (Nielson & Nielson, 1999)).

A flow analysis F for P such that F is valid is in $CallString_\beta^P$, where β is a mapping from $Labs_P \times MemEnv_F$ into Mem_F , iff whenever $\Phi_F((l_2, me), ac)$ is defined it equals $\{\beta(l, me)\}$ where l is such that⁹ $e_1 @_l ue_2^{l_2} \in SubExpr_P$. All k -CFA analyses fit into this category: for 0-CFA we take $Mem_F = \{\bullet\}$ and $\beta(l, me) = \bullet$; for 1-CFA we take $Mem_F = Labs_P$ and $\beta(l, me) = l$; and for 2-CFA (the generalization to $k > 2$ is immediate) we take $Mem_F = Labs_P \cup (Labs_P \times Labs_P)$ and define $\beta(l, me)$ as follows: let it be l if $me = []$, 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 11

The flow analysis F_1 is a 1-CFA, whereas the flow analysis F_2 is not in $CallString_\beta^{P_2}$ for any β (since Φ_{F_2} in one case returns a doubleton).

A flow analysis F for P such that F is valid is in $ArgBased_\alpha^P$, where α is a total mapping from Mem_F into $\mathcal{P}(UnAnnFlowVal_F)$, iff for all non-recursive 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 (Agesen, 1995) or the type-directed approach of (Jagannathan *et al.*, 1997); 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 $Disj_\alpha$; and if α satisfies that each element in $UnAnnFlowVal_F$ occurs in at least one $\alpha(m)$ we write $Cover_\alpha$.

Example 12

The flow analysis F_1 is in $ArgBased_\alpha^{P_1}$, with $\alpha(0) = \alpha(2) = \{ac_x\}$ and $\alpha(1) = \{Int\}$.

The flow analysis F_2 is in $ArgBased_\alpha^{P_2}$, with $\alpha(x) = \alpha(2) = \{ac_x\}$ and $\alpha(y) = \alpha(3) = \{ac_y\}$ and $\alpha(1) = \{Int\}$.

Note that by appropriate renaming (collapsing) of mementoes, both flow analyses can be converted so as to fit into a class $ArgBased_\alpha$ with $Disj_\alpha$.

4.4 Existence of Least Analyses

Given a program P , it turns out that for all β the class $CallString_\beta^P$, and for certain kinds of α also the class $ArgBased_\alpha^P$, contains a least (i.e., most precise) flow analysis; here the ordering on flow analyses is defined pointwise¹⁰ on \mathcal{C}_F , ρ_F and

⁹ It is tempting to write “ $\Phi_F((l, me), ac_0)$ ” in Fig. 4 (thus replacing l_2 by l), but then subject reduction for flows would not hold.

¹⁰ Unlike (Jagannathan *et al.*, 1997), 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) $Mem_{F_1} = Mem_{F_2}$; (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.

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-singleton¹¹.

Theorem 4.4

For all P and β the class $CallString_\beta^P$ contains a least flow analysis; and for all P and α with $Disj_\alpha$ the class $ArgBased_\alpha^P$ contains a least flow analysis provided it is not empty—a sufficient condition for which is that $Cover_\alpha$.

Proof

The theorem is an immediate consequence of Lemmas C.5 and C.6, stated and proved in Appendix C. \square

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 it on the surface is quite different from ours.

To be more precise: with R a “finitary F-analysis” in the framework of P&P, our goal is to construct a corresponding flow analysis F . As the values in P&P may be infinite whereas in our framework all flow values are finite, this may seem hard. This is where the “finitary” condition, which by the way is necessary for P&P’s translation into types, comes to rescue: it amounts to the existence of a finite set W of (possibly infinite) flow values such that all values occurring (possibly deeply nested) in R are members of W . Accordingly, the basic idea of our construction is to define Mem_F such that there exists a bijective mapping γ from $\mathcal{P}(W)$ to Mem_F . This mapping in the obvious way induces an injective mapping γ_{me} from $FlowEnv(E)$ to $MemEnv_F$ which in turn induces an injective mapping γ_V from $ValSet(E)$ to $FlowSet_F$; here $\rho \in FlowEnv(E)$ if ρ is a P&P abstract environment occurring in R , and $s \in ValSet(E)$ if s is a set of P&P flow values occurring in R . We then stipulate

$$\begin{aligned} \rho_F(x, m) &= \gamma_V(\gamma^{-1}(m)) \\ \mathcal{C}_F(l, me) &= \gamma_V(\cap\{s \mid \exists \rho \text{ with } \gamma_{me}(\rho) = me : (\rho, ue^l, s) \in R\}) \\ (\mathcal{C}_F(l, me) &= \perp \text{ if there exists no such } s) \end{aligned}$$

We can then prove that F satisfies a specification “almost” similar to the one given in Fig. 4. Clearly F belongs to $ArgBased_\alpha^E$, with $\alpha(m) = \epsilon_V(\gamma_V(\gamma^{-1}(m)))$.

The framework of P&P can model 0-CFA, by requiring the “cache” to be a singleton and to depend only on the function body. By the construction sketched above, such an analysis is translated into a flow analysis where $\Phi_F((l, me), (fn, me_0))$ is a singleton depending on fn 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).

¹¹ The “culprit” is the condition for [app] “ $\forall v \in \mathcal{C}_F(l_2, me). \exists m \in M. \{v\} \leq_V \rho_F(x, m)$ ”.

One can also go the other direction: convert a flow analysis F in $ArgBased_\alpha^E$ into a finitary F-analysis R . The trick is to use α to write mutually recursive functions δ_{me} , mapping from $MemEnv_F$ to $FlowEnv(E)$, and δ_V , mapping from $FlowSet_F$ to $ValSet(E)$. Then we stipulate

$$R = \{(\delta_{me}(me), ue^l, \delta_V(\mathcal{C}_F(l, me))) \mid (ue^l, me) \in FlowConf_F \wedge ue^l \in SubExpr_E\}$$

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 $SubExpr_P$, and suppose that $\rho_F(x, m) = \perp$ for all $m \in Mem_F$. 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 encompass¹² all entries of \mathcal{C}_F and ρ_F that are “reachable” from the root of P . Let $Analyzes_m^F(\mu f.\lambda x.ue_0^{l_0}, me)$ be a shorthand for $\mathcal{C}_F(l_0, me[f, x \mapsto m]) \neq \perp$ and $\rho_F(x, m) \neq \perp$ and $\{((\mu f.\lambda x.ue_0^{l_0}, me), Mem_F)\} \leq_V \rho_F(f, m)$. We define $Reach_P^F$ as the least set satisfying:

$$\begin{aligned} [\text{prg}] \quad & (P, []) \in Reach_P^F \\ [\text{fun}] \quad & \left((\mu f.\lambda^l x.ue_0^{l_0}, me) \in Reach_P^F \wedge Analyzes_m^F(\mu f.\lambda x.ue_0^{l_0}, me) \right) \Rightarrow \\ & \{(ue_0^{l_0}, me[f, x \mapsto m]), (x, m), (f, m)\} \subseteq Reach_P^F \\ [\text{app}] \quad & (e_1 @_l e_2, me) \in Reach_P^F \Rightarrow \{(e_1, me), (e_2, me)\} \subseteq Reach_P^F \\ [\text{suc}] \quad & (\text{succ}^l e_1, me) \in Reach_P^F \Rightarrow (e_1, me) \in Reach_P^F \\ [\text{if}] \quad & (\text{if } 0^l e_0 \text{ then } e_1 \text{ else } e_2, me) \in Reach_P^F \Rightarrow \\ & \{(e_0, me), (e_1, me), (e_2, me)\} \subseteq Reach_P^F \end{aligned}$$

Example 13

It is easy to verify that for $ue^l \in SubExpr_{P_1}$ it holds that $\mathcal{C}_{F_1}(l, me) \neq \perp$ iff $(ue^l, me) \in Reach_{P_1}^{F_1}$, and that $\rho_{F_1}(z, m) \neq \perp$ iff $(z, m) \in Reach_{P_1}^{F_1}$. Similarly for P_2 and F_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 C, shows that for reachability implies definedness provided that the flow analysis is valid.

Lemma 4.5

Let F be a flow analysis for P such that F is valid. If $(ue^l, me) \in Reach_P^F$ then (i) $\mathcal{C}_F(l, me) \neq \perp$ 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 $\rho_F(z, m) \neq \perp$.

¹² This is somewhat similar to the reachability predicate of (Gasser *et al.*, 1997).

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 $FlowSet_F$:

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

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 (Heintze, 1995), 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 14

With terminology as in Examples 4 and 8, it is easy to see that $\mathcal{F}_{T_1}(u'_x) = \{v'_x\}$ and that $\mathcal{F}_{T_1}(u_x) = \{v_x\}$.

Lemma 5.1

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 $q_{\text{int}} \in \text{dom}(u)$, so also $q_{\text{int}} \in \text{dom}(u')$ 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 $q \in \text{dom}(u)$ such that $K = \text{dom}(u.q)$ and such that a judgement for $\mu f. \lambda x. e$ occurs in D_T with address me and is derived by $[\text{fun}]^{(q:t)}$ where $t \leq_t u.q$. Since $u \leq_u u'$ we have $u.q \leq_t u'.q$, implying

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

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

Lemma 5.2

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

Definition 5.3

With T a typing for P , the flow analysis $F = \mathcal{F}(T)$ is given by $(P, IT_T, \mathcal{C}_F, \rho_F, \Phi_F)$, 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 $[\text{fun}]^{(q:t)}$,
 a judgement for $e_1 @ ue_2^{l_2}$ at me derived by $[\text{app}]^{w^\circledast}$ where $w^\circledast(q) = M$.

Example 15

It is easy to check that $F_1 = \mathcal{F}(T_1)$, and that $F_2 = \mathcal{F}(T_2)$.

Theorem 5.4

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

- F is valid and safe
- $(ue^l, me) \in \text{Reach}_P^F$ iff $\mathcal{C}_F(l, me) \neq \perp$ (for $ue \in \text{SubExpr}_P$)
- $(z, m) \in \text{Reach}_P^F$ iff $\rho_F(z, m) \neq \perp$

Proof

The result follows from Lemmas D.2, D.3, D.4, D.5, and D.6 that are all established in Appendix D. 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 $\text{AbsClos}_F \cup \{q_{\text{int}}\}$. 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:

$$\begin{aligned} \mathcal{T}_F(V) &= \bigvee_{v \in V} \{q_v : t_v\} \text{ where} \\ &\text{if } v = \text{Int} \text{ then } q_v = q_{\text{int}} \text{ and } t_v = \mathbf{int} \\ &\text{if } v = (ac, M) \text{ with } ac = (\mu f. \lambda x. e_0^{l_0}, me) \\ &\quad \text{then } q_v = ac \\ &\quad \text{and } t_v = \bigwedge_{m \in M_0} \{\{m\} : \mathcal{T}_F(\rho_F(x, m)) \rightarrow \mathcal{T}_F(\mathcal{C}_F(l_0, me[f, x \mapsto m]))\} \\ &\quad \text{where } M_0 = \{m \in M \mid \text{Analyzes}_m^F(ac)\}. \end{aligned}$$

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 16

With terminology as in Examples 4 and 8, it is easy to see—provided that q_x is considered another name for ac_x —first that $\mathcal{T}_{F_1}(\{v'_x\}) = u'_x$, and then that $\mathcal{T}_{F_1}(\{v_x\}) = u_x$ since $\mathcal{T}_{F_1}(\{v_x\}).q_x$ can be found as

$$\begin{aligned} &\bigwedge(\{1\} : \mathcal{T}_{F_1}(\rho_{F_1}(x, 1)) \rightarrow \mathcal{T}_{F_1}(\mathcal{C}_{F_1}(7, me_{x1})), \{2\} : \mathcal{T}_{F_1}(\rho_{F_1}(x, 2)) \rightarrow \mathcal{T}_{F_1}(\mathcal{C}_{F_1}(7, me_{x2}))) \\ &= \bigwedge(\{1\} : \mathcal{T}_{F_1}(\{\text{Int}\}) \rightarrow \mathcal{T}_{F_1}(\{\text{Int}\}), \{2\} : \mathcal{T}_{F_1}(\{v'_x\}) \rightarrow \mathcal{T}_{F_1}(\{v'_x\})) \\ &= \bigwedge(\{1\} : u_{\text{int}} \rightarrow u_{\text{int}}, \{2\} : u'_x \rightarrow u'_x). \end{aligned}$$

Note that without the M component in a flow value (ac, M) , v_x would equal v'_x causing $\mathcal{T}_{F_1}(\{v_x\})$ to be an infinite type (as in P&P).

Lemma 6.1

The function \mathcal{T}_F is monotone.

Proof

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

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

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

Next assume that $q \neq q_{\text{int}}$; 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 $q \in \text{dom}(\mathcal{T}_F(V'))$. For each $m \in M$ there exists u_m and u'_m such that

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

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

Lemma 6.2

$\mathcal{T}_F(\{\text{Int}\}) = u_{\text{int}}$.

For z and m such that $(z, m) \in \text{Reach}_P^F$, we define $\mathcal{T}_F^p(z, m)$ as $\mathcal{T}_F(\rho_F(z, m))$ (by Lemma 4.5 this is well-defined). And for $e = ue^l$ and me such that $(e, me) \in \text{Reach}_P^F$, we construct a judgement $\mathcal{T}_F^j(e, me)$ as

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

where $\mathcal{T}_F^A(me)$ is defined recursively by $\mathcal{T}_F^A([\])= [\]$ and $\mathcal{T}_F^A(me[z \mapsto m]) = \mathcal{T}_F^A(me)[z \mapsto \mathcal{T}_F^p(z, m)]$ (by Lemma 4.5 also this is well-defined).

Definition 6.3

With F a flow analysis for P , the typing $T = \mathcal{T}(F)$ is given by $(P, \text{Mem}_F, \text{AbsClos}_F \cup \{q_{\text{int}}\}, D_T)$, where D_T is defined by stipulating that whenever (e, me) is in Reach_P^F then D_T contains $\mathcal{T}_F^j(e, me)$, and that $\mathcal{T}_F^j(e', me')$ is a premise of $\mathcal{T}_F^j(e, me)$ iff $(e, me) \in \text{Reach}_P^F$ is among the immediate conditions (cf. the definition of Reach_P^F) for $(e', me') \in \text{Reach}_P^F$.

Example 17

It is easy to check that $\mathbb{T}_1 = \mathcal{T}(F_1)$ and that $\mathbb{T}_2 = \mathcal{T}(F_2)$, modulo renaming of the U-tags.

Clearly D_T is a tree-formed derivation, and $\mathcal{T}_F^j(e, me)$ 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. 1. This is the core of the following theorem, proved in Appendix E.

Theorem 6.4

If F is valid and safe then $T = \mathcal{T}(F)$ as constructed by Definition 6.3 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 $[\text{fun}]^{w^\lambda}$ 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 $[\text{app}]^{w^\otimes}$ where for all $q \in \text{dom}(u_1)$ it holds that $w^\otimes(q) = \Phi_F((l_2, me), q)$.

Finally, T is uniform with A_T given by \mathcal{T}^ρ

6.1 Call strings: the Type Analogy

It may be interesting¹³ to investigate what the “type counterpart” of a flow analysis F in $\text{CallString}_\beta^P$ looks like. Theorem 6.4 tells us that whenever $[\text{app}]^{w^\otimes}$ is applied in $\mathcal{T}(F)$ to produce a judgement at address ke , then (using the terminology of Fig. 1) for all q in $\text{dom}(u_1)$ it will hold that $u_1.q \leq_t \bigwedge(\beta(l, ke) : u_2 \rightarrow u)$.

In the case where F is a 1-CFA flow analysis, the corresponding typing rule becomes particularly simple:

$$\frac{A \vdash e_1 : u_1 \quad A \vdash e_2 : u_2}{A \vdash e_1 @_l e_2 : u} \quad \text{if } \forall q \in \text{dom}(u_1). u_1.q \leq_t \bigwedge(\{l\} : u_2 \rightarrow u)$$

Thus one is forced to pick the l 'th component of the elementary type $u_1.q$, even though picking some other component may yield a more precise type, as would be the case if say $u_1 = \bigvee(q : \bigwedge(\{l\} : u'_2 \rightarrow u, \{l'\} : u_2 \rightarrow u'))$ where $u_2 \leq_u u'_2$ and $u' \leq_u u$ (strict inclusions).

The above observation may give a clue to understanding why k -CFA, while conceptually simple and easily to implement, has a reputation of performing badly in practice (Wright & Jagannathan, 1998).

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 5.4 and 6.4). 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 18

Exs. 15 and 17 illustrate that $\mathcal{F} \circ \mathcal{T}$ is the identity on F_1 and F_2 , and that $\mathcal{T} \circ \mathcal{F}$ is the identity (modulo renaming of U-tags) on T_1 and T_2 . In particular $\mathcal{T} \circ \mathcal{F}$ does not necessarily introduce infinite types, thus solving an open problem in P&P.

¹³ Actually, this issue was the original motivation for our research.

7.1 Round Trips from the Flow World

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

Theorem 7.1

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

- $Reach_{P'}^{F'} = Reach_P^F$
- $\mathcal{C}_{F'}(l, me) \neq \perp$ iff $\mathcal{C}_F(l, me) \neq \perp$ and $(ue^l, me) \in Reach_P^F$, in which case $\mathcal{C}_{F'}(l, me) = \text{filter}_P^F(\mathcal{C}_F(l, me))$
- $\rho_{F'}(z, m) \neq \perp$ iff $\rho_F(z, m) \neq \perp$ and $(z, m) \in Reach_P^F$, in which case $\rho_{F'}(z, m) = \text{filter}_P^F(\rho_F(z, m))$
- $\Phi_{F'}((l_2, me), ac) = K$ 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 $\text{filter}_P^F(V)$ is given by

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

Corollary 7.2

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 19

Consider the program P given by

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

and let

$$\begin{aligned} ac_x &= (\lambda \mathbf{x}. \mathbf{x}^2, []) & v_x &= (ac_x, \{\bullet\}) \\ ac_y &= (\lambda \mathbf{y}. \lambda^4 \mathbf{z}. \mathbf{z}^5, []) & v_y &= (ac_y, \{\bullet\}) \\ ac_z &= (\lambda \mathbf{z}. \mathbf{z}^5, [\mathbf{y} \mapsto \bullet]) & v_z &= (ac_z, \{\bullet\}) \end{aligned}$$

By Theorem 4.4 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:

$$\begin{aligned} \{v_y\} &= \mathcal{C}_F(0, []) = \mathcal{C}_F(2, [\mathbf{x} \mapsto \bullet]) = \mathcal{C}_F(3, []) = \rho_F(\mathbf{x}, \bullet) \\ \{v_x\} &= \mathcal{C}_F(1, []) \end{aligned}$$

The typing $T = \mathcal{T}(F)$ contains the derivation depicted in Fig. 5, where

$$\frac{\frac{[x \mapsto u_y] \vdash x^2 : u_y}{[] \vdash \lambda^1 x.x^2 : u_x} \quad [] \vdash \lambda^3 y.\lambda^4 z.z^5 : u_y}{[] \vdash (\lambda^1 x.x^2) @_0 (\lambda^3 y.\lambda^4 z.z^5) : u_y}$$

Fig. 5. Example 19: the derivation D_T .

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

Fig. 6. Example 19: the derivation D_{T_z} .

$$\begin{aligned} u_y &= \mathcal{T}_F(\{v_y\}) = \bigvee(ac_y : \bigwedge()) \\ u_x &= \mathcal{T}_F(\{v_x\}) = \bigvee(ac_x : \bigwedge(\{\bullet\} : u_y \rightarrow u_y)) \end{aligned}$$

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

Next consider a 0-CFA flow analysis F_z where some junk that is *not* reachable has been added: F_z is as F except that

$$\begin{aligned} \mathcal{C}_{F_z}(1, []) &= \{v_x, v_z\} \\ \rho_{F_z}(z, \bullet) &= \mathcal{C}_{F_z}(5, [z \mapsto \bullet]) = \{v_y\}. \end{aligned}$$

The typing $T_z = \mathcal{T}(F_z)$ contains the derivation depicted in Fig. 6, where

$$\begin{aligned} u_y &= \mathcal{T}_{F_z}(\{v_y\}) = \bigvee(ac_y : \bigwedge()) \\ u_{xz} &= \mathcal{T}_{F_z}(\{v_x, v_z\}) = \bigvee(ac_x : \bigwedge(\{\bullet\} : u_y \rightarrow u_y), ac_z : \bigwedge(\{\bullet\} : u_y \rightarrow u_y)) \end{aligned}$$

Now it is easy to see that $\mathcal{F}_{T_z}(u_y) = \{v_y\}$ and $\mathcal{F}_{T_z}(u_{xz}) = \{v_x\}$, implying that $\mathcal{F}(\mathcal{T}(F_z)) = F$. This illustrates that $\mathcal{F} \circ \mathcal{T}$ removes junk that is not reachable.

Finally consider a 0-CFA flow analysis F_y where some junk that *is* reachable has been added: F_y is given by the entries below:

$$\begin{aligned} \{v_y\} &= \mathcal{C}_{F_y}(2, [x \mapsto \bullet]) = \mathcal{C}_{F_y}(3, []) = \rho_{F_y}(x, \bullet) = \rho_{F_y}(y, \bullet) \\ \{v_x, v_y\} &= \mathcal{C}_{F_y}(1, []) \\ \{v_z\} &= \mathcal{C}_{F_y}(4, [y \mapsto \bullet]) \\ \{v_y, v_z\} &= \mathcal{C}_{F_y}(0, []) \end{aligned}$$

The typing $T_y = \mathcal{T}(F_y)$ contains the derivation depicted in Fig. 7, where

$$\begin{aligned} u_z &= \mathcal{T}_{F_y}(\{v_z\}) = \bigvee(ac_z : \bigwedge()) \\ u_y &= \mathcal{T}_{F_y}(\{v_y\}) = \bigvee(ac_y : \bigwedge(\{\bullet\} : u_y \rightarrow u_z)) \\ u_{xy} &= \mathcal{T}_{F_y}(\{v_x, v_y\}) = \bigvee(ac_x : \bigwedge(\{\bullet\} : u_y \rightarrow u_y), ac_y : \bigwedge(\{\bullet\} : u_y \rightarrow u_z)) \\ u_{yz} &= \mathcal{T}_{F_y}(\{v_y, v_z\}) = \bigvee(ac_y : \bigwedge(\{\bullet\} : u_y \rightarrow u_z), ac_z : \bigwedge()) \end{aligned}$$

Now it is easy to see that

$$\begin{aligned} \mathcal{F}_{T_y}(u_z) &= \{v_z\} \\ \mathcal{F}_{T_y}(u_y) &= \{v_y\} \\ \mathcal{F}_{T_y}(u_{xy}) &= \{v_x, v_y\} \\ \mathcal{F}_{T_y}(u_{yz}) &= \{v_y, v_z\} \end{aligned}$$

$$\frac{\frac{[\mathbf{x} \mapsto u_y] \vdash \mathbf{x}^2 : u_y}{[] \vdash \lambda^1 \mathbf{x} . \mathbf{x}^2 : u_{xy}} \quad \frac{[\mathbf{y} \mapsto u_y] \vdash \lambda^4 \mathbf{z} . \mathbf{z}^5 : u_z}{[] \vdash \lambda^3 \mathbf{y} . \lambda^4 \mathbf{z} . \mathbf{z}^5 : u_y}}{[] \vdash (\lambda^1 \mathbf{x} . \mathbf{x}^2) @_0 (\lambda^3 \mathbf{y} . \lambda^4 \mathbf{z} . \mathbf{z}^5) : u_{yz}}$$

Fig. 7. Example 19: the derivation D_{T_y} .

implying that $\mathcal{F}(\mathcal{T}(F_y)) = F_y$. 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

$$\mathcal{T}(\mathcal{F}(\mathcal{T}(F_z))) = \mathcal{T}(F) = T \neq T_z = \mathcal{T}(F_z).$$

7.2 Round Trips from the Type World

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

Definition 7.3

A typing T is strongly consistent iff for all u that occur in¹⁴ D_T and for all $q \in \text{dom}(u)$ with $q \neq q_{\text{int}}$ the following holds:

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

Here \leq_{\wedge}^c is a subrelation of \leq_t , defined by stipulating that $\text{int} \leq_{\wedge}^c \text{int}$ and that

$$\bigwedge_{i \in I} \{K_i : u_i \rightarrow u'_i\} \leq_{\wedge}^c \bigwedge_{i \in I_0} \{K_i : u_i \rightarrow u'_i\} \text{ iff } I_0 \subseteq I.$$

Theorem 7.4

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 F. \square

Example 20

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

$$(\lambda^1 \mathbf{x} . \text{succ } \mathbf{x}) @ ((\lambda^2 \mathbf{y} . \mathbf{y}) @ 3)$$

was given essentially the typing $(P, \{\bullet\}, \{\bullet\}, D_T)$ with D_T as depicted in Fig. 8; we use $i \rightarrow i$ as a shorthand for $\bigwedge(\{\bullet\} : u_{\text{int}} \rightarrow u_{\text{int}})$. 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 .

$$\frac{\frac{[x \mapsto u_{\text{int}}] \vdash x : u_{\text{int}}}{[x \mapsto u_{\text{int}}] \vdash \text{succ } x : u_{\text{int}}} \quad \frac{[y \mapsto u_{\text{int}}] \vdash y : u_{\text{int}}}{[] \vdash \lambda y. y : \bigvee(\bullet : i \rightarrow i)} \quad \frac{[\text{fun}]^{(\bullet : i \rightarrow i)} \quad [] \vdash 3 : u_{\text{int}}}{[] \vdash (\lambda^2 y. y) @ 3 : u_{\text{int}}}}{[] \vdash \lambda x. \text{succ } x : \bigvee(\bullet : i \rightarrow i)} \quad \frac{[\text{fun}]^{(\bullet : i \rightarrow i)} \quad [] \vdash 3 : u_{\text{int}}}{[] \vdash (\lambda^1 x. \text{succ } x) @ ((\lambda^2 y. y) @ 3) : u_{\text{int}}}$$

Fig. 8. Example 20: the derivation D_T .

$$\frac{\frac{[x \mapsto u_{\text{int}}] \vdash x : u_{\text{int}}}{[x \mapsto u_{\text{int}}] \vdash \text{succ } x : u_{\text{int}}} \quad \frac{[y \mapsto u_{\text{int}}] \vdash y : u_{\text{int}}}{[] \vdash \lambda y. y : u} \quad \frac{[\text{fun}]^{(q_2 : i \rightarrow i)} \quad [] \vdash 3 : u_{\text{int}}}{[] \vdash (\lambda y. y) @ 3 : u_{\text{int}}}}{[] \vdash \lambda x. \text{succ } x : u} \quad \frac{[\text{fun}]^{(q_1 : i \rightarrow i)} \quad [] \vdash (\lambda y. y) @ 3 : u_{\text{int}}}{[] \vdash (\lambda x. \text{succ } x) @ ((\lambda y. y) @ 3) : u_{\text{int}}}$$

Fig. 9. Example 20: the derivation $D_{T'}$.

$$\mathcal{C}_F(1, []) = \mathcal{C}_F(2, []) = \mathcal{F}_T(\bigvee(\bullet : i \rightarrow i)) = \{((\lambda x. \text{succ } x, []), \{\bullet\}), ((\lambda y. y, []), \{\bullet\})\}$$

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

Then the typing $T' = \mathcal{T}(F)$ is given by $(P, \{\bullet\}, \{q_1, q_2\}, D_{T'})$, with $q_1 = (\lambda x. \text{succ } x, [])$ and $q_2 = (\lambda y. y, [])$, and with $D_{T'}$ as depicted in Fig. 9 where u denotes $\bigvee(q_1 : i \rightarrow i, q_2 : i \rightarrow i)$. Note that T' is strongly consistent, as guaranteed by Theorem 7.4.

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 (Wright & Jagannathan, 1998), 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 (Wells *et al.*, 1997; Dimock *et al.*, 1997). 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.

A Semantics

The following small-step semantics rules are basically adapted from N&N; but we deviate from N&N in that we do not truncate the semantic environment se in the rule (fun) (cf. the rule $[fun]$ in Fig. 4). The semantics satisfies the following property: if $se \vdash e \Rightarrow e'$, then (i) $e \in \mathit{Exps}_P$ implies $e' \in \mathit{Exps}_P$; and (ii) if $e = ue^l$, then there exists a ue' such that $e' = ue'^l$.

$$\begin{array}{l}
(var) \quad se \vdash z^l \Rightarrow sv^l \quad \text{if } sv = se(z) \\
(fun) \quad se \vdash \mu f. \lambda^l x. e \Rightarrow \mathbf{close}^l \mu f. \lambda x. e \text{ in } se \\
(app_l) \quad \frac{se \vdash e_1 \Rightarrow e'_1}{se \vdash e_1 @_l e_2 \Rightarrow e'_1 @_l e_2} \\
(app_r) \quad \frac{se \vdash e_2 \Rightarrow e'_2}{se \vdash e_1 @_l e_2 \Rightarrow e_1 @_l e'_2} \\
(app_v) \quad \frac{se \vdash (\mathbf{close}^{l_1} fn_1 \text{ in } se_1) @_l sv_2^{l_2}}{\Rightarrow \mathbf{bind}^l se_1 [f \mapsto (\mathbf{close} fn_1 \text{ in } se_1), x \mapsto sv_2] \text{ in } e_1} \\
\quad \text{where } fn_1 = \mu f. \lambda x. e_1 \\
(succ) \quad \frac{se \vdash e_1 \Rightarrow e'_1}{se \vdash \mathbf{succ}^l e_1 \Rightarrow \mathbf{succ}^l e'_1} \\
(succ_v) \quad se \vdash \mathbf{succ}^l c^{l_0} \Rightarrow c_1^l \quad , \text{ if } c_1 = c + 1 \\
(if) \quad \frac{se \vdash e_0 \Rightarrow e'_0}{se \vdash \mathbf{if}0^l e_0 \text{ then } e_1 \text{ else } e_2 \Rightarrow \mathbf{if}0^l e'_0 \text{ then } e_1 \text{ else } e_2} \\
(if_0) \quad se \vdash \mathbf{if}0^l 0^{l_0} \text{ then } ue_1^{l_1} \text{ else } ue_2^{l_2} \Rightarrow ue_1^l \\
(if_{>}) \quad se \vdash \mathbf{if}0^l c^{l_0} \text{ then } ue_1^{l_1} \text{ else } ue_2^{l_2} \Rightarrow ue_2^l, \text{ if } c \neq 0 \\
(bind) \quad \frac{se_1 \vdash e_1 \Rightarrow e'_1}{se \vdash \mathbf{bind}^l se_1 \text{ in } e_1 \Rightarrow \mathbf{bind}^l se_1 \text{ in } e'_1} \\
(bind_v) \quad se \vdash \mathbf{bind}^l se_1 \text{ in } sv_1^{l_1} \Rightarrow sv_1^l
\end{array}$$

B Types

Proof of Lemma 3.1

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

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

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

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

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

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

$$\begin{aligned} K_i &= \bigcup_{i_0 \in I_0} K_{i_0} \text{ and } \forall i_0 \in I_0. u'_{i_0} Q_u u'_i \text{ and} \\ \forall q \in \text{dom}(u_i). \exists i_0 \in I_0. q \in \text{dom}(u_{i_0}) \text{ and } u_i.q Q_t u_{i_0}.q. \end{aligned}$$

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

$$\begin{aligned} K'_{i'} &= \bigcup_{j \in J} K''_j \text{ and } \forall j \in J. u''_{1j} \leq_u u'_{1i'} \text{ and} \\ \forall q \in \text{dom}(u'_{0i'}). \exists j \in J. q \in \text{dom}(u''_{0j}) \text{ and } u'_{0i'}.q \leq_t u''_{0j}.q. \end{aligned}$$

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

$$\begin{aligned} K''_j &= \bigcup_{i \in I_j} K_i \text{ and } \forall i \in I_j. u_{1i} \leq_u u''_{1j} \text{ and} \\ \forall q_j \in \text{dom}(u''_{0j}). \exists i \in I_j. q_j \in \text{dom}(u_{0i}) \text{ and } u''_{0j}.q_j \leq_t u_{0i}.q_j. \end{aligned}$$

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

$$\begin{aligned} K'_{i'} &= \bigcup_{i \in I_0} K_i \text{ and } \forall i \in I_0. u_{1i} Q_u u'_{1i'} \text{ and} \\ \forall q \in \text{dom}(u'_{0i'}). \exists j \in J. \exists i \in I_j. q \in \text{dom}(u_{0i}) \text{ and } u'_{0i'}.q \leq_t u''_{0j}.q \leq_t u_{0i}.q \end{aligned} \quad (3)$$

where the latter line implies that

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

Now (3) and (4) demonstrates the second half of (2). $\square \square$

Proof of Lemma 3.2

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

Lemma B.1

If $A \vdash e : u$ and $u \leq_u u'$ then also $A \vdash e : u'$.

Lemma B.2

Assume that for all $z \in FV(e)$ it holds that $A(z) = A'(z)$. Then $A \vdash e : u$ implies $A' \vdash e : u$.

Lemma B.3

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

Lemma B.4

If $A \vdash sv : u$ then there exists $q \in \text{dom}(u)$ such that $A \vdash sv : \bigvee(q : u.q)$.

We are now ready to prove Theorem 3.2, 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 $se \vdash e_1 \Rightarrow e'_1$. 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 $[\text{app}]^w$ 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 $[\] \vdash sv : A(z)$, and that $A \vdash z : u$, implying $A(z) \leq_u u$. By Lemmas B.1 and B.2 this demonstrates the desired judgement $A \vdash sv : u$.

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

(succ_v). The situation is that $se \vdash \text{succ}^l c^{l0} \Rightarrow c_1^l$. Our assumption is that $A \vdash \text{succ}^l c^{l0} : u$, implying $u_{\text{int}} \leq_u u$. Then clearly also $A \vdash c_1 : u$, as desired.

(if₀). (the case (if_>) is similar.) The situation is, with $e = \text{if}0^l 0^{l0} \text{ then } ue_1^{l1} \text{ else } e_2$, that $se \vdash e \Rightarrow ue_1^l$. Among the premises of the judgement $A \vdash e : u$ is a judgement of the form $A \vdash ue_1^{l1} : u_1$, where $u_1 \leq_u u$. By Lemma B.1, this shows that $A \vdash ue_1^l : u$ as desired.

(bind). The situation is that $se \vdash \text{bind}^l se_1 \text{ in } e_1 \Rightarrow \text{bind}^l se_1 \text{ in } e'_1$ because $se_1 \vdash e_1 \Rightarrow e'_1$. Our assumptions are that $A \vdash \text{bind}^l se_1 \text{ in } e_1 : u$, 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 $A \vdash \text{bind}^l se_1 \text{ in } e'_1 : u$, as desired.

(bind_v). The situation is that $se \vdash \text{bind}^l se_1 \text{ in } sv^{l1} \Rightarrow sv^l$. Our assumptions are that $A \vdash \text{bind}^l se_1 \text{ in } sv^{l1} : u$, implying that there exists A_1 and $u_1 \leq_u u$ such

that $A_1 \vdash sv : u_1$. By Lemmas B.1 and B.2 (recall that $FV(sv) = \emptyset$) we infer the desired judgement $A \vdash sv : u$.

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

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

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

$$A \vdash sv_1 : u_1 \tag{5}$$

$$A \vdash sv_2 : u_2 \tag{6}$$

employing that

$$\forall q_1 \in \mathit{dom}(u_1). u_1.q_1 \leq_t \bigwedge (w^\circledast(q_1) : u_2 \rightarrow u). \tag{7}$$

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

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

such that this judgement is derived using $[\mathbf{fun}]^{(q:t)}$. 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 B.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 B.1 and B.2 infer

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

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

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

(7) and (9) implies $t \leq_t \bigwedge (w^\circledast(q) : u_2 \rightarrow u)$ 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 B.1 and B.2) that

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

(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

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

This completes the proof. \square

C Flows

Proof of Lemma 4.2

As a preparation, we state a number of rather trivial¹⁵ facts:

Lemma C.1

If $sv \mathcal{V}_F V$ and $V \leq_V V'$ then also $sv \mathcal{V}_F V'$.

Lemma C.2

If $sv \mathcal{V}_F V$ then there exists $v \in V$ such that $sv \mathcal{V}_F \{v\}$.

Lemma C.3

$sv \mathcal{V}_F \mathcal{C}_F(l, me)$ iff $F \models^{me} sv^l$.

Lemma C.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 4.2, 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

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

(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 $F \models^{me} e'_1 @_l e_2$. 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:

(var). 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 C.1 this shows $sv \mathcal{V}_F \mathcal{C}_F(l, me)$, which by Lemma C.3 amounts to the desired $F \models^{me} sv^l$.

(fun). The situation is that $se \vdash \mu f. \lambda^l x. e \Rightarrow \mathbf{close}^l fn \mathbf{in} se$, 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

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

amounts to $\{((fn, me), Mem_F)\} \leq_V \mathcal{C}_F(l, me)$. This demonstrates that $F \models^{me} \text{close}^l fn \text{ in } se$, as desired.

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

(if₀). (the case (if_>) is similar.) The situation is that

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

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

(bind). The situation is that $se \vdash \text{bind}^l se_1 \text{ in } e_1 \Rightarrow \text{bind}^l se_1 \text{ in } e'_1$ 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 $F \models^{me} \text{bind}^l se_1 \text{ in } e_1$, 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 $F \models^{me} \text{bind}^l se_1 \text{ in } e'_1$, as desired.

(bind_v). The situation is that $se \vdash \text{bind}^l se_1 \text{ in } sv^{l_1} \Rightarrow sv^l$. By our assumption $F \models^{me} \text{bind}^l se_1 \text{ in } sv^{l_1}$ 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 C.3 amounts to $sv \mathcal{V}_F \mathcal{C}_F(l_1, me_1)$. By Lemma C.1 we infer $sv \mathcal{V}_F \mathcal{C}_F(l, me)$, which by Lemma C.3 amounts to the desired $F \models^{me} sv^l$.

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

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

By assumption we have

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

from which we infer that $F \models^{me} sv_1^{l_1}$ and $F \models^{me} sv_2^{l_2}$, which by Lemma C.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 C.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 C.1 implies

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

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

$$F \models^{me_1[f, x \mapsto m]} e_0 \quad (9)$$

$$\mathcal{C}_F(l_0, me_1[f, x \mapsto m]) \leq_V \mathcal{C}_F(l, me) \quad (10)$$

$$\{((fn_1, me_1), Mem_F)\} \leq_V \rho_F(f, m). \quad (11)$$

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 $F \models^{me} \text{bind}^l se_1[f \mapsto sv_1, x \mapsto sv_2]$ in e_0 , as desired.

This completes the proof. \square

Lemma C.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 $Labs_P \times MemEnv_F$ into Mem_F we can ensure that F belongs to $CallString_\beta^P$;
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 $(e, me) \in \mathcal{G}_F(X)$.

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

$e = \mu f. \lambda^l x. e_0$. 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 $(e, me) \in \mathcal{G}_F(X)$.

$e = e_1 @_l e_2$. 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 $\mathcal{C}_F(l, me) \neq \perp$ 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 $fn \in Funs_P$ implying $e_0 \in SubExpr_P$, and that $FV(fn) \subseteq dom(me_0)$. This clearly shows that $M' \subseteq M_0$ and that for all $m \in M'$ we have $(e_0, me_0[f, x \mapsto m]) \in X$, $\mathcal{C}_F(l_0, me_0[f, x \mapsto m]) \leq_V \mathcal{C}_F(l, me)$, $\rho_F(x, m) \neq \perp$, 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

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

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. \square

Lemma C.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 $CallString_\beta^P$;
2. there exists α with $Disj_\alpha$ such that all F_j belong to $ArgBased_\alpha^P$.

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

Then F is valid. In case 1, F will belong to $CallString_\beta^P$; in case 2, F will belong to $CallString_\alpha^P$.

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 $\rho_F(x, m) \neq \perp$ then for all $j \in J$ also $\rho_{F_j}(x, m) \neq \perp$ 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 $(P, []) \in X$, and that if $\mathcal{C}_F(l, me) \neq \perp$ with $e = ue^l \in SubExpr_P$ and $(e, me) \in FlowConf_F$ then for all $j \in J$ it holds that $\mathcal{C}_{F_j}(l, me) \neq \perp$ 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 $\perp \neq \rho_{F_j}(z, me(z)) \leq_V \mathcal{C}_{F_j}(l, me)$. Then clearly $\perp \neq \rho_F(z, me(z)) \leq_V \mathcal{C}_F(l, me)$, 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 $\{((\mu f. \lambda x. e_0), me), Mem_{F_j}\} \leq_V \mathcal{C}_{F_j}(l, me)$ 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

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

which (since for $i = 1, 2$ we have $(e_i, me) \in FlowConf_F$ and $e_i \in SubExpr_P$) shows that

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

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

$$e_0 \in SubExpr_P \text{ and } \forall m \in Mem_F. FV(e_0) \subseteq dom(me_0[f, x \mapsto m]). \tag{15}$$

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

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

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

$$\begin{aligned} \forall j \in J. \forall m \in M_j. F_j \models^{me_0[f, x \mapsto m]} e_0 \\ \mathcal{C}_{F_j}(l_0, me_0[f, x \mapsto m]) \leq_V \mathcal{C}_{F_j}(l, me) \\ \rho_{F_j}(x, m) \neq \perp \wedge \{(ac_0, Mem_F)\} \leq_V \rho_{F_j}(f, m) \end{aligned} \tag{18}$$

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

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

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

$$\begin{aligned} \forall m \in M. (e_0, me_0[f, x \mapsto m]) \in X \\ \mathcal{C}_F(l_0, me_0[f, x \mapsto m]) \leq_V \mathcal{C}_F(l, me) \\ \rho_F(x, m) \neq \perp \wedge \{(ac_0, Mem_F)\} \leq_V \rho_F(f, m) \end{aligned} \tag{20}$$

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

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

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

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

It will be sufficient to show that

$$\exists m. \forall j \in J. m_j = m. \quad (23)$$

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 $F_j \in \text{ArgBased}_\alpha^P$ 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.

□

Proof of Lemma 4.5

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

[prg]. Here $e = P$ and $me = []$. Since F is valid we have $F \models [] P$, 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 $z \in \{x, f\}$, $(z, m) \in \text{Reach}_P^F$ is part of the conclusion of [fun]. Otherwise, $(z \mapsto m) \in me'$ so $(z, m) \in \text{Reach}_P^F$ follows from the induction hypothesis applied to $(\mu f. \lambda x. e, me') \in \text{Reach}_P^F$.

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

The remaining cases. They are similar to [app]. □

D Type to Flow

First an auxiliary result:

Lemma D.1

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

Proof

Let $k \in \text{dom}(t)$ 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 $[\text{fun}]^{(q:t)}$ ensures that $\bigvee(q : t) \leq_u u'_k$. Clearly $((\mu f.\lambda x.e, ke), \text{dom}(t))$ belongs to $\mathcal{F}_T(\bigvee(q : t))$, so by Lemma 5.1 we infer

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

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

Lemma D.2

It holds that F is valid.

Proof

We define

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

Note that if $e = ue^l \in \text{SubExpr}_P$ and $\mathcal{C}_F(l, me) \neq \perp$ then $(e, me) \in X$; our task can thus be accomplished (as trivially $(P, []) \in X$) 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 5.1 we infer

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

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

$e = \mu f.\lambda^l x.e_0$. Let J be derived using $[\text{fun}]^{(q:t)}$; then $\bigvee(q : t) \leq_u u$. With $ac = (\mu f.\lambda x.e_0, me)$ we infer that $(ac, \text{dom}(t)) \in \mathcal{F}_T(\bigvee(q : t))$, so Lemma 5.1 yields

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

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

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

$$u_1.q \leq_t \bigwedge (w^\textcircled{!}(q) : u_2 \rightarrow u). \quad (1)$$

We first observe that

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

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 $[\text{fun}]^{(q_0:t_0)}$, with $q_0 \in \text{dom}(u_1)$ and

$$t_0 \leq_t u_1.q_0 \quad (3)$$

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

$$\Phi_F((l_2, me), ac_0) = M \quad (4)$$

and (1) tells us that

$$M \subseteq M_0. \quad (5)$$

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

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

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

$$(e_0, me_0[f, x \mapsto m]) \in X \quad (7)$$

$$\mathcal{C}_F(l_0, me[f, x \mapsto m]) = \mathcal{F}_T(u'_m) \leq_V \mathcal{F}_T(u) \leq_V \mathcal{C}_F(l, me) \quad (8)$$

$$\rho_F(x, m) \neq \perp \quad (9)$$

$$\{(ac_0, Mem_F)\} \leq_V \rho_F(f, m) \quad (10)$$

where (8) follows from Lemma 5.1 since (6) implies $u'_m \leq_u u$, and where (10) is provided by Lemma D.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 $q \in \text{dom}(u_2)$ 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 5.1 implies the desired relation

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

$e = c^l$. Since $u_{\text{int}} \leq_u u$ we by Lemmas 5.1 and 5.2 infer that

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

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

$e = \text{succ}^l e_1$. Since $u_{\text{int}} \leq_u u$ we by Lemmas 5.1 and 5.2 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 $(e, me) \in \mathcal{G}_F(X)$.

$e = \text{if}0^l e_0 \text{ then } e_1 \text{ else } e_2$. 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 5.1 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 D.3

It holds that F is safe.

Proof

Let $ue^l \in \text{SubExpr}_P$; we must consider several cases.

$ue = ue_1^{l_1} @ e_2$. 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 $[\text{app}]^{w^\circ}$. If $\text{Int} \in \mathcal{C}_F(l_1, me)$ then $q_{\text{int}} \in \text{dom}(u_1)$, but by the side condition of applying $[\text{app}]^{w^\circ}$ this is impossible.

$ue = \text{succ } ue_1^{l_1}$. 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 $[\text{suc}]$. Therefore $u_1 \leq_u u_{\text{int}}$, which by Lemmas 5.1 and 5.2 implies that $\mathcal{F}_T(u_1) \leq_V \mathcal{F}_T(u_{\text{int}}) = \{\text{Int}\}$. This shows that if $v \in \mathcal{C}_F(l_1, me)$ then $v = \text{Int}$.

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

Lemma D.4

Let $e = ue^l \in \text{SubExpr}_P$. If $\mathcal{C}_F(l, me) \neq \perp$ then $(e, me) \in \text{Reach}_P^F$.

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 $me = []$ 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 $(e_0, me_0) \in \text{Reach}_P^F$. Therefore $(e, me) \in \text{Reach}_P^F$ 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 $(e, me) \in \text{Reach}_P^F$, thanks to Lemma D.1. \square

Lemma D.5

If $\rho_F(z, m) \neq \perp$ then $(z, m) \in \text{Reach}_P^F$.

Proof

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

Lemma D.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 $\rho_F(z, m) \neq \perp$.

Proof

From Lemma D.2 we know that F is valid, so the claim follows from Lemma 4.5.

□

E Flow to Type

Proof of Theorem 6.4

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 4.5) then (as F is valid) it holds that $F \models^{me} e$.

$ue = z$. Since $F \models^{me} e$ we have $\perp \neq \rho_F(z, me(z)) \leq_V \mathcal{C}_F(l, me)$ which by Lemma 6.1 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).

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

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

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

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

Since $F \models^{me} e$ we have $\{v\} \leq_V \mathcal{C}_F(l, me)$, so Lemma 6.1 tells us that

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

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

$ue = e_1 @ e_2$. 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 $q_0 \in \text{dom}(\mathcal{T}_F(\mathcal{C}_F(l_1, me)))$ be given. In order to show that J is derived from its premises using $[\text{app}]^{w^\circledast}$ where w^\circledast is as in the theorem, we must show

$$\begin{aligned} & \mathcal{T}_F(\mathcal{C}_F(l_1, me)).q_0 \\ \leq_t & \bigwedge (\Phi_F((l_2, me), q_0) : \mathcal{T}_F(\mathcal{C}_F(l_2, me)) \rightarrow \mathcal{T}_F(\mathcal{C}_F(l, me))). \end{aligned} \quad (1)$$

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

$$\begin{aligned} \mathcal{T}_F(\mathcal{C}_F(l_1, me)).q_0 &= \bigwedge_{m \in M_0} \{ \{m\} : \mathcal{T}_F(\rho_F(x, m)) \rightarrow \mathcal{T}_F(\mathcal{C}_F(l_0, me_0[f, x \mapsto m])) \} \\ &\text{ where } M_0 = \{m \in M_1 \mid \text{Analyzes}_m^F(ac_0)\}. \end{aligned}$$

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

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

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 6.1 implies the desired relation.

Concerning (3), we for $q \in \text{dom}(\mathcal{T}_F(\mathcal{C}_F(l_2, me)))$ 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 6.1 implies $\mathcal{T}_F(\{v\}) \leq_u \mathcal{T}_F(\rho_F(x, m))$ and hence the desired relation.

$ue = c$. From $F \models^{me} e$ we infer $\{\text{Int}\} \leq_V \mathcal{C}_F(l, me)$ which by Lemmas 6.1 and 6.2 implies $u_{\text{int}} \leq_u \mathcal{T}_F(\mathcal{C}_F(l, me))$, showing that J is derivable from its (empty) set of premises.

$ue = \text{succ } e_1$. 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 $\{\text{Int}\} \leq_V \mathcal{C}_F(l, me)$ and since F is safe we have $\mathcal{C}_F(l_1, me) \leq_V \{\text{Int}\}$. By Lemmas 6.1 and 6.2 this implies

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

which shows that J is derivable from its premises.

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

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

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 $\mathcal{C}_F(l_0, me) \leq_V \{\text{Int}\}$. By Lemmas 6.1 and 6.2 this implies

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

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 $(z, m) \in \text{Reach}_P^F$ so there exists $\mu f.\lambda x.e$ and me with $z \in \{f, x\}$ such that $(e, me[f, x \mapsto m]) \in \text{Reach}_P^F$; this shows that D_T in fact does contain a judgement with address $me[f, x \mapsto m]$.

□

F Round Trips

Proof of Theorem 7.1

First note that $T = \mathcal{T}_F$ is uniform (by Theorem 6.4), so $F' = \mathcal{F}(\mathcal{T}(F))$ is valid and safe (by Theorem 5.4). Clearly $\text{Mem}_{F'} = IT_T = \text{Mem}_F$.

For the claim that $\text{Reach}_P^{F'} = \text{Reach}_P^F$, we observe (by Theorems 5.4 and 6.4 and by Definitions 5.3 and 6.3) that $(e, me) \in \text{Reach}_P^{F'}$ iff (with $e = ue^l$) $\mathcal{C}_{F'}(l, me) \neq \perp$ iff D_T contains a judgement for e with address me iff $(e, me) \in \text{Reach}_P^F$, and that $(z, m) \in \text{Reach}_P^{F'}$ iff $\rho_{F'}(z, m) \neq \perp$ iff $A_T(z, m)$ is defined iff $\mathcal{T}_F^\rho(z, m)$ is defined iff $(z, m) \in \text{Reach}_P^F$.

We next establish an auxiliary result:

Lemma F.1

For $V \in \text{FlowSet}_F$ it holds that $\mathcal{F}_T(\mathcal{T}_F(V)) = \text{filter}_P^F(V)$.

Proof

Clearly $\text{Int} \in \mathcal{F}_T(\mathcal{T}_F(V))$ iff $q_{\text{int}} \in \text{dom}(\mathcal{T}_F(V))$ iff $\text{Int} \in V$. 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 $q \in \text{dom}(\mathcal{T}_F(V))$ such that $M' = \text{dom}(\mathcal{T}_F(V).q)$ and such that a judgement for $\mu f.\lambda x.e$ occurs in D_T with address me and is derived by $[\text{fun}]^{w^\lambda}$ where w^λ takes the form $(q : t)$; by Theorem 6.4 we infer that $q = ac$. Since $ac \in \text{dom}(\mathcal{T}_F(V))$ we next infer that there exists M such that $(ac, M) \in V$ and such that $M' = \text{dom}(\mathcal{T}_F(V).ac)$ is given by

$$\{m \in M \mid \text{Analyzes}_m^F(ac)\}$$

which since $(\mu f.\lambda x.e, me) \in \text{Reach}_P^F$ (by Definition 6.3) implies that

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

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

Next assume that $(ac, M') \in \text{filter}_P^F(V)$, which amounts to $(\mu f.\lambda x.e, me) \in \text{Reach}_P^F$ and the existence of M such that $(ac, M) \in V$ and such that $M' = \{m \in M \mid (e, me[f, x \mapsto m]) \in \text{Reach}_P^F\}$. Then $ac \in \text{dom}(\mathcal{T}_F(V))$ and clearly $M' = \text{dom}(\mathcal{T}_F(V).ac)$, and (by Theorem 6.4) D_T contains a judgement for $\mu f.\lambda x.e$ at address me derived by $[\text{fun}]^{w^\lambda}$ with w^λ of the form $(ac : \mathcal{T}_F(\{(ac, \text{Mem}_F)\}).ac)$ where by Lemma 6.1 we have $\mathcal{T}_F(\{(ac, \text{Mem}_F)\}) \leq_u \mathcal{T}_F(V)$ and hence $\mathcal{T}_F(\{(ac, \text{Mem}_F)\}).ac \leq_t \mathcal{T}_F(V).ac$. 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 F.1 we infer as desired that $\mathcal{C}_{F'}(l, me) = V$ iff $(ue^l, me) \in Reach_P^F$ and $V = \mathcal{F}_T(\mathcal{T}_F(\mathcal{C}_F(l, me))) = \text{filter}_P^F(\mathcal{C}_F(l, me))$.

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

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 $[\text{fun}]^{w^\lambda}$ with w^λ of the form $(q : t)$; (ii) a judgement for e with address me derived by $[\text{app}]^{w^\circledast}$ where $w^\circledast(q) = K$. From the former judgement, we infer by Theorem 6.4 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 $ac \in \text{dom}(\mathcal{T}_F(\mathcal{C}_F(l_1, me)))$ and by Theorem 6.4 that $w^\circledast(ac) = \Phi_F((l_2, me), ac)$. 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 6.4 we infer that D_T contains at address me_0 a judgement for $\mu f. \lambda x. e_0$ derived by $[\text{fun}]^{w^\lambda}$ with w^λ of the form $(ac : t)$; and (since $ac \in \text{dom}(\mathcal{T}_F(\mathcal{C}_F(l_1, me)))$) 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 $[\text{app}]^{w^\circledast}$ with $w^\circledast(ac) = K$. This demonstrates that $\Phi_{F'}((l_2, me), ac) = K$. \square

Proof of Corollary 7.2

In the following, we shall apply Theorem 7.1 to F as well as to F' .

First we observe that $Mem_{F''} = Mem_{F'}$, that $Reach_P^{F'} = Reach_P^F$ and therefore $\text{filter}_P^{F'} = \text{filter}_P^F$, 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 $\mathcal{C}_{F''}(l, me) = \text{filter}_P^{F'}(\mathcal{C}_{F'}(l, me)) = \text{filter}_P^F(\text{filter}_P^F(\mathcal{C}_F(l, me))) = \text{filter}_P^F(\mathcal{C}_F(l, me)) = \mathcal{C}_{F'}(l, me)$. 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 7.1, we infer that $\mathcal{C}_{F'}(l_1, me) \neq \perp$ and that $\mathcal{C}_{F'}(l_1, me) = \text{filter}_P^F(\mathcal{C}_F(l_1, me))$, 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 7.1 on F') that $\Phi_{F''}((l_2, me), ac)$ is defined.

\square

Proof of Theorem 7.4

Let $F = \mathcal{F}(T)$, so that $T' = \mathcal{T}(F)$. First note that F is valid and safe (by Theorem 5.4) so T' is uniform (by Theorem 6.4). 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 5.4) $\mathcal{C}_F(l, me) \neq \perp$ 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 6.3 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 $FlowVal_F$ 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 6.4, this judgement is derived by $[\text{fun}]^{w^\lambda}$ where $w^\lambda = (ac : \mathcal{T}_F(\{(ac, Mem_F)\}).ac)$. It is immediate from the definition of \mathcal{T}_F that

$$\mathcal{T}_F(\{(ac, Mem_F)\}).ac \leq_{\wedge}^c \mathcal{T}_F(V).ac = \mathcal{T}_F(\{(ac, M)\}).ac$$

which shows the ‘‘at least one’’ part of Definition 7.3; 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 $[\text{fun}]^{(q:t)}$ 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 $A' = \mathcal{T}_F^A(me)$. To see that A' equals A , note that

$$\begin{aligned} \mathcal{T}_F^A([\]) &= [\] \text{ and } \mathcal{T}_F^A(me[z \mapsto m]) = \mathcal{T}_F^A(me)[z \mapsto \mathcal{T}_F^\rho(z, m)] = \mathcal{T}_F^A(me)[z \mapsto \\ &\mathcal{T}_F(\mathcal{F}_T(A_T(z, m)))] = \mathcal{T}_F^A(me)[z \mapsto A_T(z, m)] \end{aligned}$$

and that A equals $A(me)$ where

$$A([\]) = [\] \text{ and } A(me[z \mapsto m]) = 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:

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

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 $[\text{fun}]^{(q:t_0)}$, where $q = (\mu f.\lambda x.e_0, me_0)$ and where $t_0 \leq_{\wedge}^c t$. The premises of this judgement are thus of the form

$$k \in K_0: A[f \mapsto u'_k, x \mapsto u_k] \vdash e_0 : u'_k \quad (3)$$

where $K \subseteq K_0$. By the “at most one” part of Definition 7.3 we infer that $\mathcal{F}_T(\bigvee(q : t)) = \{(q, K)\}$, so with $e_0 = ue_0^{l_0}$ we have

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

where K' is given by

$$\{k \in K \mid \text{Analyzes}_k^F(q)\}.$$

The judgements in (3) demonstrate (using Lemma D.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). \square

References

- Agesen, Ole. (1995). The Cartesian product algorithm. *Pages 2–26 of: Proceedings of ecoop'95, seventh european conference on object-oriented programming*, vol. 952. Springer-Verlag.
- Amadio, Roberto, & Cardelli, Luca. (1993). Subtyping recursive types. *Acm trans. on prog. langs. and systs.*, **15**(4), 575–631.
- Banerjee, Anindya. (1997). A modular, polyvariant, and type-based closure analysis. *In: (ICFP '97, 1997)*.
- Dimock, Allyn, Muller, Robert, Turbak, Franklyn, & Wells, J. B. (1997). Strongly typed flow-directed representation transformations. *In: (ICFP '97, 1997)*.
- Gasser, Kirsten L. Solberg, Nielson, Flemming, & Nielson, Hanne Riis. (1997). Systematic realisation of control flow analyses for CML. *In: (ICFP '97, 1997)*.
- Heintze, Nevin. (1995). Control-flow analysis and type systems. *In: (SAS '95, 1995)*.
- ICFP '97. (1997). *Proc. 1997 int'l conf. functional programming*. ACM Press.
- Jagannathan, Suresh, & Weeks, Stephen. (1995). A unified treatment of flow analysis in higher-order languages. *Pages 393–407 of: Conf. rec. 22nd ann. ACM symp. princ. of prog. langs.*
- Jagannathan, Suresh, Weeks, Stephen, & Wright, Andrew. (1997). Type-directed flow analysis for typed intermediate languages. *Proc. 4th int'l static analysis symp.* LNCS, vol. 1302. Springer-Verlag.
- Nielson, Flemming, & Nielson, Hanne Riis. (1997). Infinitary control flow analysis: A collecting semantics for closure analysis. *Pages 332–345 of: Conf. rec. popl '97: 24th ACM symp. princ. of prog. langs.*
- Nielson, Flemming, & Nielson, Hanne Riis. (1999). Interprocedural control flow analysis. *Pages 20–39 of: Proc. european symp. on programming*. LNCS, vol. 1576. Springer-Verlag.
- Palsberg, Jens, & O'Keefe, Patrick. (1995). A type system equivalent to flow analysis. *Acm trans. on prog. langs. and systs.*, **17**(4), 576–599.
- Palsberg, Jens, & Pavlopoulou, Christina. (1998). From polyvariant flow information to intersection and union types. *Pages 197–208 of: Conf. rec. popl '98: 25th ACM symp. princ. of prog. langs.* Superseded by (Palsberg & Pavlopoulou, 1999).
- Palsberg, Jens, & Pavlopoulou, Christina. 1999 (Feb.). *From polyvariant flow information to intersection and union types*. A substantially revised version of (Palsberg & Pavlopoulou, 1998). Available at <http://www.cs.purdue.edu/homes/palsberg/paper/popl98.ps.gz>.

- SAS '95. (1995). *Proc. 2nd int'l static analysis symp.* LNCS, vol. 983.
- Schmidt, David. (1995). Natural-semantics-based abstract interpretation. *In: (SAS '95, 1995).*
- Shivers, Olin. (1991). *Control flow analysis of higher order languages.* Ph.D. thesis, Carnegie Mellon University.
- Wells, J. B., Dimock, Allyn, Muller, Robert, & Turbak, Franklyn. (1997). A typed intermediate language for flow-directed compilation. *Pages 757–771 of: Proc. 7th int'l joint conf. theory & practice of software development.*
- Wright, Andrew, & Jagannathan, Suresh. (1998). Polymorphic splitting: An effective polyvariant flow analysis. *Acm trans. on prog. langs. and systs.*, **20**, 166–207.