Type Inference with Expansion Variables and Intersection Types in System E and an Exact Correspondence with β-Reduction*

Sébastien Carlier J. B. Wells

ABSTRACT

System E is a recently designed type system for the λ-calculus with intersection types and *expansion variables*. During automatic type inference, expansion variables allow postponing decisions about which non-syntax-driven typing rules to use until the right information is available and allow implementing the choices via substitution.

This paper uses expansion variables in a unification-based automatic type inference algorithm for System E that succeeds for every β-normalizable λ-term. We have implemented and tested our algorithm and released our implementation publicly. Each step of our unification algorithm corresponds to exactly one β-reduction step, and *vice versa*. This formally verifies and makes precise a correspondence between type inference and β-reduction that previously was only known informally. This also shows that type inference with intersection types and expansion variables can, in effect, carry out an arbitrary amount of partial evaluation of the program being analyzed.

1. DISCUSSION

1.1 Background and Motivation

1.1.1 Types for Programs and Type Inference

Types have been used extensively to analyze computer program properties without executing the programs, for purposes such as detecting programming errors, enforcing abstract inter-module interfaces, justifying compiler optimizations, and enforcing security properties.

In the *type assignment* style, a type system associates each untyped (i.e., free of type annotations) term (e.g., computer program fragment) with 0 or more *typings*, where each typing is a pair of a result type and a type environment for free term variables. *Type inference* is finding a typing to assign to an untyped term, if possible. Type inference provides the benefits of types while relieving programmers from having to manually supply the types.

Type inference algorithms generally solve constraints, often by *unification*, the process of computing a *solution* (if one exists) assigning values to variables so that constraints become solved. Unification-based type inference is usually understandable and often efficiently implementable. The most widely used type inference algorithm is Milner's W for the well known Hindley/Milner system [27], used in languages such as SML, OCaml, and Haskell.

The amount of polymorphism offered by these programming languages can be obtained more simply by reducing all `let` expressions (a form of partial evaluation) and then doing type inference using only simple types. However, compiler writers do not use this technique for two reasons: (1) it works only when the whole program is given at once to the compiler, but programmers demand at least separate compilation; (2) it is very inefficient, because definitions are re-analyzed every time they are used. Forall-quantified types, as used in the Hindley/Milner system, allow separate compilation (provided module interfaces are known) with simple and efficient type inference. However, they are somewhat inflexible and do not support *compositional analysis*, i.e., analysing modules without knowledge about other modules.

1.1.2 Intersection Types

Intersection types were introduced for the λ-calculus by Coppo and Dezani [8] and independently by Pottinger [29]. In both cases, a major motivation was the connection between β-reduction and intersection types, which makes intersection types well suited for program analysis.

Intersection type systems have been developed for many kinds of program analysis (e.g., flow [1], strictness [18], deadcode [13, 10], and totality [7]) usable for justifying compiler optimizations to produce better machine code. Intersection types seem to have the potential to be a general, flexible framework for many program analyses.

In most intersection type systems, every typable term has a *principal typing* [37], one that is stronger than all other typings assignable to that term. Principal typings improve the possibilities of analyzing programs incrementally and minimizing reanalysis cost after program modifications.

1.1.3 Type Inference for Full Intersection Types

The first type inference algorithms for intersection type systems were by Coppo, Dezani and Venneri [9], and Ronchi

*Partially supported by EC FP5 grant IST-2001-33477, EPSRC grant GR/L 41545/01, NATO grant CRG 971607, NSF grant 0113193 (ITR), and Sun Microsystems equipment grant EDUD-7826-990410-US.

della Rocca and Venneri [32]. These algorithms give principal typings for β-normal forms, and it is separately proven that a principal typing for a term's β-normal form is also principal for the term. This approach is unattractive because it needs terms to be reduced to β-normal form before finding types for them. Another disadvantage is that getting other typings from principal typings in this approach uses operations such as *substitution*, *expansion*, and *lifting*, and the older expansion definitions are hard to understand.

Ronchi della Rocca [31] devised the first unification-based type inference algorithm for a non-rank-restricted intersection type system. This algorithm yields principal typings when it succeeds. Because typability for the full system is undecidable, the algorithm is of course sometimes non-terminating; to ensure termination, one can restrict type height. This algorithm uses the older, complicated definition of expansion, which has several undesirable implications: (1) the unification procedure requires global knowledge in addition to the two types to unify, which make the technique delicate to extend; (2) expansion is only defined on types and typings, but not on typing derivations, which are thus not straightforward to obtain as a product of type inference (typing derivations are needed for use in compiler optimizations); (3) the older notion of expansion only allows intersection introduction in argument position, and is thus unsuitable for accurate resource tracking under call-by-need and call-by-value evaluation.

Regnier [30] presents an inference algorithm for an intersection type system called *De*. System *De* is similar to that of [31], with the essential addition of *labels* in types to carry out an operation like expansion. Using *nets*, an untyped version of proof nets from Linear Logic [14], Regnier shows an exact correspondence between net reduction in Linear Logic and type inference in System *De*. Boudol and Zimmer [4] give another inference algorithm that uses neither expansion nor labels, but attaches a set of type variables to each typing constraint. This approach appears to be inflexible and hard to generalize beyond pure λ-calculus. Both the approach of Regnier and that of Boudol and Zimmer are stated to be unification-based, but neither approach produces *unifiers* that solve sets of unification constraints, which makes them both difficult to use in practical settings.

1.1.4 Rank-2 Intersection Types

Leivant [23] introduced a rank-2 intersection type system and remarked that it is similar in typing power to Hindley/Milner. In fact, rank-2 intersection type systems type strictly more terms than Hindley/Milner, although the difference is not important.

Van Bakel [35] gave the first unification-based type inference algorithm for a rank-2 intersection type system. Jim [19] studied extensions of rank-2 intersection type systems with practical programming features and discussed the importance of principal typings.

Damiani has recently made further progress with rank-2 intersection types. In one paper [11], Damiani adds support for conditionals and letrec (mutually recursive bindings) as well as support for a let-binding mechanism that effectively adds some of the power of one more rank to obtain some of the expressiveness of rank-3 intersection types. In another paper [12], Damiani works on symmetric linking of modules. This work is limited to rank-2 intersection types and can not use the full potential of intersection types to analyze programs as precisely as needed.

1.1.5 Expansion Variables

System I [20, 22] had the first practical type inference algorithm for a full, non-rank-restricted intersection type system. System I introduced *expansion variables* and a single operation integrating expansion and substitution; together, these features vastly simplified expansion.

Unfortunately, System I has several technical limitations. The substitution operation (which contains expansion) has a built-in renaming of type and expansion variables (needed for complete type inference) that prevents substitutions from being composable. An awkward workaround for this problem called *safe composition* was developed, but safe composition is hard to understand and people were error-prone when working with it. System I has other technical limitations such as non-associative and non-commutative intersections and no weakening; together, these limitations prevent it from having subject reduction.

The recently developed System E [6] improves in many ways on System I. The full System E (only a fraction of its power is needed for this paper) is more flexible than the extremely flexible intersection type system of Barendregt, Coppo, and Dezani [2]. Contrary to Van Bakel's advice to make intersection type systems as lean as possible [36], System E does not restrict where intersection type constructors (and thus also expansion variables) can be used. Flexible expansion variable placement allows expansion variables to establish namespaces. In turn, this allows a substantially simpler way of integrating expansion and substitution. In particular, System E does not need the automatic built-in fresh-renaming done during expansion in System I. As a result, substitution and expansion in System E are more robust than they are in System I and composition works for substitutions and expansions. These improvements make it much simpler to design type inference methods for System E, leading to the new results in this paper.

1.2 Summary of Contributions

1. We present a clear and precise unification-based intersection type inference algorithm in System E and prove that it succeeds in typing any β-normalizing λ-term. Our presentation is easier to understand and implement than previous presentations. A strength of our result is that our algorithm uses only the !-free fragment of System E restricted to trivial subtyping. (Our algorithm can work unchanged with ! and non-trivial subtyping.)
2. Our intersection type inference approach provides a simple and clean step-for-step *β-reduction/unification correspondence*. In the main algorithm phase, each step of our unify-β rule on constraints corresponds to exactly one β-reduction step on λ-terms, and *vice versa*. Any β-reduction strategy can be simulated by a strategy of using unify-β, and *vice versa*. Our proof that our algorithm succeeds for all β-normalizing λ-terms takes advantage of this correspondence and uses the well known β-normalizing leftmost/outermost strategy. Our correspondence improves over that of Regnier [30] both by being more formal and also by being direct instead of being the composition of two correpondences with untyped *nets* as an intermediate notion. This correspondence is impossible for systems without the flexibility of intersection types, because they can not type even all strongly

β-normalizing terms [34].
The clear presentation of this correspondence helps to share the understanding that any intersection type inference approach will be equivalent to *partial evaluation* followed by a monovariant analysis. Intersection type inference in effect can do an arbitrary amount of partial evaluation via type unification, i.e., do the equivalent of any amount of β-reduction of the analyzed term.

3. Our intersection type inference approach is the only one to provide a notion of *readback* that allows extracting β-reduced λ-terms from partially solved constraints during type inference. This makes it easy to prove the step-for-step β-reduction/unification correspondence and makes the correspondence easier to understand. The use of readback makes our approach clearer than the approaches of Boudol and Zimmer [4] and also Regnier [30].
4. We have implemented all of the algorithms described in this paper and made them available for downloading (`http://www.macs.hw.ac.uk/DART/software/system-e/`) and also via a web interface (`http://www.macs.hw.ac.uk/ultra/compositional-analysis/system-e/`).
5. Our intersection type inference approach has many advantages over competing approaches because it is formulated inside System E.
   (a) Expansion in System E is guided by E-variables. As a result, expansion operations can be built and used without global knowledge of the relevant typings.
   (b) Unlike nearly all other intersection type systems, System E has *skeletons*, compact syntactic representations of typing derivations. Skeletons allow stating and proving many properties more clearly and precisely.
   (c) Because expansion is defined on all of the mathematical entities in System E, including skeletons, it is easy to get different forms of output such as a full typing derivation or just a typing (result type and type environment). Constructing typing derivations is vital for use in compilers with typed intermediate representations. In contrast, competing intersection type inference approaches fail to adequately document how to construct typing derivations; this knowledge is either deeply buried in proofs or simply omitted.
   (d) Unlike expansion in all other previous intersection type systems, expansion in System E has no restrictions on where it can occur. Expansion in System E can splice in intersection constructors in type positions that are to the right of arrows and in skeleton positions that are not application arguments. The first major benefit of this feature (used in this paper) is that analysis polyvariance can be managed by a scheme of properly arranging the nesting of a finite number of E-variables. This is simpler than renaming schemes used in other approaches and greatly eases implementation.
   (e) Not only does System E have expansion and substitution integrated in a single operation (like System I), but the composition of expansions is straightforward, making it easy to express constraint solving solutions.
6. Future work extending the algorithms in this paper will have additional advantages due to being formulated inside System E.
   (a) A further benefit of System E's location-unrestricted expansion is that type inference will be able to do resource-aware analysis of call-by-need and call-by-value evaluation, rather than being applicable only to call-by-name evaluation (which is unused in practice).
   (b) The metatheory of System E has been worked out for a larger system than presented in this paper; this larger system has the ! type constructor. In combination with the step-for-step β-reduction/unification correspondence, type inference will be able to obtain polyvariance by doing the equivalent of any amount of partial evaluation of the analyzed program followed by a cruder, more traditional monovariant analysis using ! and subtyping to collapse the analysis. Because of the careful way ! has been integrated into the full System E, the analysis results will be precisely marked as to where the information is exact and where it is approximate. This ability for type inference to partially evaluate leads to the potential of analysis that simultaneously is compositional and has easily adjustable cost and precision. (The algorithm presented in this paper always has exact precision and the same cost as normalization.) In contrast, the widely used algorithm W for Hindley/Milner is non-compositional and does the equivalent of a fixed amount of partial evaluation (unfolding all let-expressions) followed by the standard (monovariant) first-order unification of simple type inference [25]; all information in the results must be assumed to be approximate. Although we leave full exploration of this promising possibility to future work, this motivation helps justify the significance of this work.

1.3 Other Future Work

Because types are often exposed to programmers, a major design goal for many type systems has been making types suitable for human comprehension. Unfortunately, this conflicts with making types suitable for accurate and flexible program analysis. Intersection types are good for accurate analysis. However, inferred intersection types may be extremely detailed and thus are likely to be unsuitable for presenting to humans. Alternative type error reporting methods such as *type error slicing* [15, 16] can avoid presenting these types directly to programmers. Investigation is needed to combine type error slicing with System E.

Module boundary interfaces are generally intended to abstract away from the actual software on either side of the interface, so that implementations can be switched. Also, module boundary interfaces must be compact and easily understandable by humans. For these reasons, $\forall$ and $\exists$ quantifiers are appropriate for use in module boundary types. An open problem is how to use very flexible and accurate types such as intersection types for analysis and then check whether they imply types using $\forall$ and $\exists$ quantifiers.

We (Carlier, Kfoury, and Wells) and also independently Mairson and Neergaard [28] have noticed a correspondence between *solving of type inference constraints* in System I and *reduction of nets*. Intersection types correspond to contraction nodes, ω (not in the original System I) to 0-ary contraction nodes (weakening), E-variables to boxes, T-variables to axiom links, and constraints to cut links. Each type inference constraint solving step in System I (using the version of the rules of [20]) closely corresponds to a net reduction step. Expansion describes net transformations, and our unification algorithm does the equivalent of cut-elimination via substitution for E-variables. In System E (but not in System I), the equalities imposed on types and constraints in sec. 3.2 correspond to the flexibility of nets.

This connection has several implications. First, System E offers a syntactic alternative to nets that is easier for precise reasoning. Second, the type inference algorithm in sec. 4 works in larger steps than the earlier algorithms for System I, and this is vital for our exact correspondence with individual β-reduction steps. Thus, our new unification/β-reduction correspondence can also indirectly give the same new result for net reduction. Third, System E could lead to a Linear Logic extension with intersection as a proof-functional connective [24] (unlike the usual truth-functional connectives). In this new system, Regnier's nets could be annotated with formulas. (Although Mairson [26] earlier suggested that net edges can be annotated with Linear Logic formulas, a counter-example, due to Urzyczyn, is the net for $22\mathsf{K}$, with $2 = \lambda f.\lambda x.f(fx)$ and $\mathsf{K} = \lambda y.\lambda z.y$.)

1.4 Other Related Work

Sayag and Mauny [33] characterize principal typings in intersection type systems and show they are isomorphic to β-normal forms. Although interesting, the correspondence is limited to normal forms and sheds no light on the relation between β-reduction and type inference.

1.5 Acknowledgements

This paper benefited from detailed comments by Adam Bakewell, Assaf Kfoury, Henning Makholm, and Jeff Polakow and from helpful discussions with Harry Mairson and Peter Neergaard on correspondences between proof nets and type inference with expansion variables.

2. PRELIMINARY DEFINITIONS

This section defines generic mathematical notions. Let i, j, m, n, p, and q range over $\{0,1,2,\ldots\}$ (the natural numbers). Let $\pi_1(\langle a,b\rangle) = a$ and $\pi_2(\langle a,b\rangle) = b$. Given a function f, let $f[a \mapsto b] = (f \setminus \{\, (a,c) \mid (a,c) \in f \,\}) \cup \{(a,b)\}$. Let r range over binary relations. Let $\xrightarrow{r}$ be alternate infix notation for r. Let $\xrightarrow{r}\!\!\!\!\!\rightarrow$ be the transitive and reflexive (w.r.t. the intended carrier set) closure of r. Let $r;r'$ be the composition of r and r', i.e., $r;r' = \{\, (a,c) \mid \exists b.r(a,b) \wedge r'(b,c) \,\}$. Given a context C, let $C[U]$ stand for C with the single occurrence of $\square$ replaced by U. For example, $(\lambda x.\square)[x] = \lambda x.x$.

If S names a set and φ is defined as a metavariable ranging over S, let S^* be the set of *sequences* over S as per the following grammar, quotiented by the subsequent equalities, and let $\vec{\varphi}$ be a metavariable ranging over S^*:

$$\vec{\varphi} \in S^* ::= \epsilon \mid \varphi \mid \vec{\varphi}_1 \cdot \vec{\varphi}_2$$
$$\epsilon \cdot \vec{\varphi} = \vec{\varphi}, \quad \vec{\varphi} \cdot \epsilon = \vec{\varphi}, \quad (\vec{\varphi}_1 \cdot \vec{\varphi}_2) \cdot \vec{\varphi}_3 = \vec{\varphi}_1 \cdot (\vec{\varphi}_2 \cdot \vec{\varphi}_3)$$

For example, $\vec{n}$ ranges over $\{0,1,2,\ldots\}^*$ (sequences of natural numbers). Length 1 sequences are equal to their sole member; this requires taking some care.

Given an order $<$ on a set X, let the *lexicographic-extension order* $<_{\mathsf{lex}}$ of $<$ be the least relation s.t. for any $\vec{x}, \vec{z} \in X^*$ and $y, y' \in X$ where $y < y'$ both $\vec{x} <_{\mathsf{lex}} \vec{x} \cdot y \cdot \vec{z}$ and $\vec{x} \cdot y \cdot \vec{y} <_{\mathsf{lex}} \vec{x} \cdot y' \cdot \vec{z}$ hold.

Diagrams illustrate formal statements. A diagram means that for all entities linked to solid lines satisfying the relations attached to the solid lines, the additional entities linked to dashed lines exist satisfying the relations attached to the dashed lines. For example, the following diagram means $\forall m,n,q.\ (m < n) \wedge (n \leq q) \Rightarrow \exists p.\ (m \leq p) \wedge (p < q)$:

$$\begin{tikzcd} m \arrow[r, "<"] \arrow[d, dashed, "\leq"'] & n \arrow[d, "\leq"] \\ p \arrow[r, dashed, "<"] & q \end{tikzcd}$$

3. SYSTEM E

This section presents System E. See [6] for full details.

3.1 Syntax

Fig. 1 defines the syntactic entities used in this paper. Note the distinction between the metavariables x, α, and e and concrete variables like x_0, a_1, and e_2. The main difference from the original System E definition [6] is that the ! operator is omitted in this paper.

We define operator precedence, including for ordinary function application ($f(a)$) and modification ($f[a \mapsto b]$), and for later-defined operations like expansion application ($[E]\,X$) and term-variable substitution ($M_1[x{:=}M_2]$). The precedence groups follow, from highest to lowest:

$$\begin{array}{ll} \text{group 1:} & Q^{:\tau},\ f(a),\ f[a \mapsto b],\ M_1[x := M_2], \\ & v := \Phi,\ C[M],\ D[Q],\ \vec{\varphi}_1 \cdot \vec{\varphi}_2 \\ \text{group 2:} & e\,X,\ [E]\,X,\ (\phi,\ S) \\ \text{group 3:} & X_1 \cap X_2,\ e/S \\ \text{group 4:} & \tau_1 \to \tau_2,\ M \mathbin{@} N,\ Q_1 \mathbin{@} Q_2,\ S_1;S_2 \\ \text{group 5:} & \tau_1 \leq \tau_2,\ \lambda x.M,\ \lambda x.Q, \end{array}$$

As examples of binding conventions, $e\,\alpha_1 \cap \alpha_2 \to \alpha_3 = ((e\,\alpha_1) \cap \alpha_2) \to \alpha_3$, and $(e\,\alpha_1 \leq \alpha_2) = ((e\,\alpha_1) \leq \alpha_2)$, and $\lambda x.\, x^{:\alpha_1} \mathbin{@} y^{:\alpha_2} = \lambda x.\,(x^{:\alpha_1} \mathbin{@} y^{:\alpha_2})$. As is usual, application is left-associative so that $M_1 \mathbin{@} M_2 \mathbin{@} M_3 = (M_1 \mathbin{@} M_2) \mathbin{@} M_3$ (similarly for skeletons) and function types are right-associative so that $\tau_1 \to \tau_2 \to \tau_3 = \tau_1 \to (\tau_2 \to \tau_3)$. Let the expression $\tau_1 \cap \cdots \cap \tau_n$ denote $\tau_1 \cap (\tau_2 \cap (\cdots \cap \tau_n))$ when $n \geq 1$ and ω when $n = 0$. Extending E-variable application to sequences, let $\epsilon\,X = X$ and $(\vec{e} \cdot e)\,X = \vec{e}\,(e\,X)$.

3.2 Equalities

Terms and skeletons are quotiented by α-conversion as usual [3], where $\lambda x.M$ and $\lambda x.\,Q$ bind the variable x.

For types and constraints, the definitions of fig. 1 are modified by imposing equalities for E-variable application, the $\cap$ operator, and the ω constant. The $\cap$ operator is associative and commutative with ω as its unit. E-variable application distributes over $\cap$ and ω. (The constant ω is a 0-ary version of $\cap$.) Formally, these rules hold:

$$\begin{array}{lll} T_1 \cap (T_2 \cap T_3) = (T_1 \cap T_2) \cap T_3 & & \\ T_1 \cap T_2 \qquad\quad = T_2 \cap T_1 & & \omega \cap T = T \\ e\,(T_1 \cap T_2) \quad\ = e\,T_1 \cap e\,T_2 & & e\,\omega \quad = \omega \end{array}$$

Both α-conversion and the additional rules for types and constraints are imposed as *equalities*, where "$=$" is mathematical equality (as it should be). For example, $\omega \cap \alpha = \alpha$. After this modification, the syntactic sorts no longer form an initial algebra, so care must be taken.

3.3 Operations on Syntax

Let $T \sqsubseteq T'$ iff $T' = T \cap T''$ and let $T \sqsupseteq T'$ iff $T' \sqsubseteq T$.

Let $M[x := N]$ and respectively $M \xrightarrow{\beta} N$ denote the usual notions [3] for untyped λ-terms of term-variable substitution and respectively β-reduction.

The sorts and their abstract syntax grammars and metavariables:

$x \in$ Term-Variable $::= \mathsf{x}_i$ $\quad M, N \in$ Term $::= x \mid \lambda x.M \mid M_1 \mathbin{@} M_2$

$\alpha \in$ T-Variable $::= \mathsf{a}_i$ $\quad C \in$ Term-Context $::= \Box \mid \lambda x.C \mid C \mathbin{@} M \mid M \mathbin{@} C$

$e \in$ E-Variable $::= \mathsf{e}_i$ $\quad \tau \in$ Type $::= \tau_1 \cap \tau_2 \mid e\,\tau \mid \omega \mid \alpha \mid \tau_1 \to \tau_2$

$\phi \in$ ET-Assignment $::= \alpha := \tau \mid e := E$ $\quad E \in$ Expansion $::= E_1 \cap E_2 \mid e\,E \mid \omega \mid S$

$S \in$ ET-Substitution $::= \boxdot \mid \phi,\ S$ $\quad \Delta \in$ Constraint $::= \Delta_1 \cap \Delta_2 \mid e\,\Delta \mid \omega \mid \tau_1 \mathbin{\dot\le} \tau_2$

$Q \in$ Skeleton $::= Q_1 \cap Q_2 \mid e\,Q \mid \omega^M \mid x^{:\tau} \mid \lambda x.\,Q \mid Q_1 \mathbin{@} Q_2 \mid Q^{:\tau}$

$D \in$ Skel-Context $::= \Box \mid D \cap Q \mid Q \cap D \mid e\,D \mid \lambda x.\,D \mid D \mathbin{@} Q \mid Q \mathbin{@} D \mid D^{:\tau}$

Metavariables ranging over subsets or multiple sorts:

$v ::= e \mid \alpha \qquad \Phi ::= E \mid \tau \qquad T ::= \tau \mid \Delta \qquad X ::= \tau \mid E \mid \Delta \mid Q \qquad Y ::= \tau \mid E \mid \Delta \qquad \dot\Delta ::= e\,\dot\Delta \mid \bar\Delta \qquad \bar\Delta ::= \tau_1 \mathbin{\dot\le} \tau_2$

Figure 1: Syntax grammars and metavariable conventions.

Let term be the least-defined function such that:

$$\begin{aligned}
\mathsf{term}(x^{:\tau}) &= x\\
\mathsf{term}(\lambda x.\,Q) &= \lambda x.\mathsf{term}(Q)\\
\mathsf{term}(Q_1 \mathbin{@} Q_2) &= \mathsf{term}(Q_1) \mathbin{@} \mathsf{term}(Q_2)\\
\mathsf{term}(Q^{:\tau}) &= \mathsf{term}(e\,Q) = \mathsf{term}(Q)\\
\mathsf{term}(\omega^M) &= M\\
\mathsf{term}(Q_1 \cap Q_2) &= \mathsf{term}(Q_1) \quad \text{if } \mathsf{term}(Q_1) = \mathsf{term}(Q_2)
\end{aligned}$$

A skeleton Q is *well formed* iff $\mathsf{term}(Q)$ is defined. For example, $Q = x^{:\tau_1} \cap y^{:\tau_2}$ is ill-formed if $x \neq y$, because $\mathsf{term}(Q)$ is undefined.

Convention 3.1. Henceforth, only well formed skeletons are considered. □

If $\dot\Delta = \vec{e}\bar\Delta$, then $\mathsf{E\text{-}path}(\dot\Delta) = \vec{e}$ is the *E-path* of $\dot\Delta$. If D is a skeleton context, we define $\mathsf{E\text{-}path}(D)$, the E-path of the context hole in D, by induction, as follows:

$$\begin{aligned}
&\mathsf{E\text{-}path}(\Box) = \epsilon\\
&\mathsf{E\text{-}path}(e\,D) = e \cdot \mathsf{E\text{-}path}(D)\\
&\mathsf{E\text{-}path}(\lambda x.\,D) = \mathsf{E\text{-}path}(D^{:\tau}) = \mathsf{E\text{-}path}(D \mathbin{@} Q)\\
&\quad = \mathsf{E\text{-}path}(Q \mathbin{@} D) = \mathsf{E\text{-}path}(D \cap Q)\\
&\quad = \mathsf{E\text{-}path}(Q \cap D) = \mathsf{E\text{-}path}(D)
\end{aligned}$$

3.4 Expansion Application

Definition 3.2 (Expansion application). Fig. 2 defines the application of substitutions to E-variables, and of expansions to types, expansions, constraints, and skeletons. □

Lemma 3.3 (Expansion application composition). *Given any E_1, E_2, X, $[[E_1]\,E_2]\,X = [E_1]\,[E_2]\,X$.* □

Let $E_1; E_2 = [E_2]\,E_1$ (composition of expansions). By lem. 3.3, the ";" operator is associative. Although $E_1; E_2$ is not much shorter than $[E_2]\,E_1$, it is easier to follow.

An assignment ϕ may stand for a substitution $(\phi,\ \boxdot)$ and is to be interpreted that way if necessary. The higher precedence of $(v := \Phi)$ over $(\phi,\ S)$ applies here. For example, as a substitution $(e_1 := e_2 := S_2,\ e_3 := S_3)$ stands for $((e_1 := (e_2 := S_2)),\ e_3 := S_3)$ which stands for $((e_1 := ((e_2 := S_2),\ \boxdot)),\ (e_3 := S_3),\ \boxdot)$.

Let e/S stand for $(e := e\,S)$. Thus, when necessary, e/S stands for $((e := e\,S),\ \boxdot)$. The "/" notation builds a substitution that affects variables underneath an E-variable, because $[e/S]\,e\,X = e\,[S]\,X$ and $[e/S]\,X = X$ if $X \neq e\,X'$. For example, $S = (\mathsf{e}_0/(\mathsf{a}_1 := \tau_1),\ \mathsf{a}_0 := \tau_0)$ stands for $S = (\mathsf{e}_0 := \mathsf{e}_0\,(\mathsf{a}_1 := \tau_1,\ \boxdot),\ \mathsf{a}_0 := \tau_0,\ \boxdot)$ and in this case it holds that $[S]\,(\mathsf{e}_0\,\mathsf{a}_1 \to \mathsf{a}_0) = \mathsf{e}_0\,\tau_1 \to \tau_0$.

We extend this notation to E-variable sequences so that $\vec{e} \cdot e/S$ stands for $\vec{e}/e/S$ and ϵ/S stands for S.

3.5 Type Environments and Typing Rules

Type environments, ranged over by A and B, are total functions from Term-Variable to Type that map only a finite number of variables to non-ω types.

Definition 3.4 (Operations on type environments).

$$\begin{aligned}
[E]\,A &= \{\, (x, [E]\,A(x)) \mid x \in \text{Term-Variable} \,\}\\
A \cap B &= \{\, (x, A(x) \cap B(x)) \mid x \in \text{Term-Variable} \,\}\\
e\,A &= \{\, (x, e\,A(x)) \mid x \in \text{Term-Variable} \,\}\\
\mathsf{env}_\omega &= \{\, (x, \omega) \mid x \in \text{Term-Variable} \,\}
\end{aligned}$$ □

Let $(x_1 : \tau_1, \cdots, x_n : \tau_n)$ be $\mathsf{env}_\omega[x_1 \mapsto \tau_1] \cdots [x_n \mapsto \tau_n]$. Observe, for every e, E_1, E_2, A, and x, that (1) $[E_1 \cap E_2]\,A = [E_1]\,A \cap [E_2]\,A$, (2) $[E]\,(A \cap B) = [E]\,A \cap [E]\,B$, (3) $[e\,\boxdot]\,A = e\,A$, (4) $(e\,A)(x) = e\,A(x)$, (5) $[\omega]\,A = \mathsf{env}_\omega$, and finally (6) $[E]\,A[x \mapsto \tau] = ([E]\,A)[x \mapsto [E]\,\tau]$.

Fig. 3 gives the typing rules of System E used in this paper. The rules derive *judgements* of the form $(M \rhd Q) : \langle A \vdash \tau \rangle \,/\, \Delta$. The pair $\langle A \vdash \tau \rangle$ of a type environment A and a *result type* τ is a *typing*. The constraint Δ is discussed below. The skeleton Q syntactically represents a tree of typing rule uses that derives a judgement for the untyped term $\mathsf{term}(Q)$. Thus, a skeleton is basically a typing derivation.

Definition 3.5 (Valid skeleton). A skeleton Q is *valid* iff there exist M, A, τ, and Δ s.t. $(M \rhd Q) : \langle A \vdash \tau \rangle \,/\, \Delta$. □

Convention 3.6. Only valid skeletons are considered. □

A skeleton Q uniquely determines all components in its judgement. Let typing, constraint, tenv, and rtype be functions s.t. $(M \rhd Q) : \langle A \vdash \tau \rangle \,/\, \Delta$ implies $\mathsf{typing}(Q) = \langle A \vdash \tau \rangle$, $\mathsf{constraint}(Q) = \Delta$, $\mathsf{tenv}(Q) = A$, and $\mathsf{rtype}(Q) = \tau$.

The intended meaning of $(M \rhd Q) : \langle A \vdash \tau \rangle / \Delta$ is that Q is a proof that M has the typing $\langle A \vdash \tau \rangle$, provided the constraint Δ is *solved* w.r.t. a subtyping relation. A typing's meaning depends on the subtyping relation. The typing rules do not check whether a constraint Δ is solved to allow (1) using different subtyping relations and (2) using the rules to generate constraints for use in type inference. For subtyping this paper uses only equality on types. Future papers will use other relations (e.g., $\le_{\mathsf{nlin}}$ and $\le_{\mathsf{flex}}$ from [6]).

Lemma 3.7 (Admissibility of expansion). *If $(M \rhd Q) : \langle A \vdash \tau \rangle \,/\, \Delta$, then $(M \rhd [E]\,Q) : \langle [E]\,A \vdash [E]\,\tau \rangle \,/\, [E]\,\Delta$.* □

A constraint Δ is *solved*, written $\mathsf{solved}(\Delta)$, iff Δ is of the form $\vec{e}_1\,(\tau_1 \mathbin{\dot\le} \tau_1) \cap \cdots \cap \vec{e}_n\,(\tau_n \mathbin{\dot\le} \tau_n)$. Given a judgement $(M \rhd Q) : \langle A \vdash \tau \rangle \,/\, \Delta$, we say that the entire judgement and its skeleton Q are *solved* iff Δ is solved. The *unsolved part* of a constraint Δ, written $\mathsf{unsolved}(\Delta)$, is the smallest constraint Δ_1 such that $\Delta = \Delta_1 \cap \Delta_2$ and $\mathsf{solved}(\Delta_2)$ for some Δ_2.

$$
\begin{array}{llllll}
[E_1 \mathbin{\dot\cap} E_2]\, X & = [E_1]\, X \mathbin{\dot\cap} [E_2]\, X & [\boxdot]\, \alpha & = \alpha & [S]\, e\, X & = [[S]\, e]\, X \\
[e\, E]\, X & = e\, [E]\, X & [\boxdot]\, e & = e\, \boxdot & & \\
[\omega]\, Y & = \omega & [v := \Phi,\ S]\, v' & = [S]\, v' \text{ if } v \neq v' & [S]\, \boxdot & = S \\
[\omega]\, Q & = \omega^{\mathsf{term}(Q)} & [v := \Phi,\ S]\, v & = \Phi & [S]\, (v := \Phi,\ S') & = (v := [S]\, \Phi,\ [S]\, S') \\
[S]\, (X_1 \mathbin{\dot\cap} X_2) & = [S]\, X_1 \mathbin{\dot\cap} [S]\, X_2 & [S]\, \omega & = \omega & [S]\, x^{:\tau} & = x^{:[S]\, \tau} \\
[S]\, (\tau_1 \mathrel{\dot\leq} \tau_2) & = [S]\, \tau_1 \mathrel{\dot\leq} [S]\, \tau_2 & [S]\, \omega^M & = \omega^M & [S]\, \lambda x.\, Q & = \lambda x.\, [S]\, Q \\
[S]\, (\tau_1 \to \tau_2) & = [S]\, \tau_1 \to [S]\, \tau_2 & [S]\, Q^{:\tau} & = ([S]\, Q)^{:[S]\, \tau} & [S]\, (Q_1 \mathbin{@} Q_2) & = [S]\, Q_1 \mathbin{@} [S]\, Q_2
\end{array}
$$

Figure 2: Expansion application.

$$
\text{(abstraction)}\ \frac{(M \triangleright Q) : \langle A \vdash \tau \rangle \ /\ \Delta}{(\lambda x.\, M \triangleright \lambda x.\, Q) : \langle A[x \mapsto \omega] \vdash A(x) \to \tau \rangle \ /\ \Delta}
\qquad
\text{(variable)}\ \frac{}{(x \triangleright x^{:\tau}) : \langle (x : \tau) \vdash \tau \rangle \ /\ \omega}
$$

$$
\text{(application)}\ \frac{(M_1 \triangleright Q_1) : \langle A_1 \vdash \tau_2 \to \tau_1 \rangle \ /\ \Delta_1; \quad (M_2 \triangleright Q_2) : \langle A_2 \vdash \tau_2 \rangle \ /\ \Delta_2}{(M_1 \mathbin{@} M_2 \triangleright Q_1 \mathbin{@} Q_2) : \langle A_1 \mathbin{\dot\cap} A_2 \vdash \tau_1 \rangle \ /\ \Delta_1 \mathbin{\dot\cap} \Delta_2}
\qquad
\text{(omega)}\ \frac{}{(M \triangleright \omega^M) : \langle \mathsf{env}_\omega \vdash \omega \rangle \ /\ \omega}
$$

$$
\text{(intersection)}\ \frac{(M \triangleright Q_1) : \langle A_1 \vdash \tau_1 \rangle \ /\ \Delta_1; \quad (M \triangleright Q_2) : \langle A_2 \vdash \tau_2 \rangle \ /\ \Delta_2}{(M \triangleright Q_1 \mathbin{\dot\cap} Q_2) : \langle A_1 \mathbin{\dot\cap} A_2 \vdash \tau_1 \mathbin{\dot\cap} \tau_2 \rangle \ /\ \Delta_1 \mathbin{\dot\cap} \Delta_2}
\qquad
\text{(E-variable)}\ \frac{(M \triangleright Q) : \langle A \vdash \tau \rangle \ /\ \Delta}{(M \triangleright e\, Q) : \langle e\, A \vdash e\, \tau \rangle \ /\ e\, \Delta}
$$

$$
\text{(result subtyping)}\ \frac{(M \triangleright Q) : \langle A \vdash \tau_1 \rangle \ /\ \Delta}{(M \triangleright Q^{:\tau_2}) : \langle A \vdash \tau_2 \rangle \ /\ \Delta \mathbin{\dot\cap} (\tau_1 \mathrel{\dot\leq} \tau_2)}
$$

Figure 3: Typing rules.

Remark 3.8 (Relation to traditional typing rules). Solved judgements and skeletons correspond respectively to traditional typing judgements and derivations. Traditional typing rules are merely the special case of the typing rules given in fig. 3 where all constraints are solved. The rules in fig. 3 can also be used to generate unsolved constraints to be solved by a type inference algorithm, as done in sec. 4. The presentation here not only saves space by combining the two roles of the rules, but also guarantees in a simple way that type inference results yield valid typing derivations. ☐

4. TYPE INFERENCE

This section presents a type inference algorithm for System E. We show how to infer a typing for any normalizing untyped λ-term M, where each constraint-solving step exactly corresponds to one β-reduction step, starting from M. We also give an algorithm that reconstructs the β-reduced term from each intermediate stage of type inference.

The approach to type inference is as follows. Given an untyped term M as input, (1) pick an initial skeleton Q such that $\mathsf{term}(Q) = M$, (2) do unification to find a substitution S solving $\mathsf{constraint}(Q)$, and (3) use Q and S to calculate a solved skeleton (typing derivation) or a typing for M.

4.1 Initial Skeletons

We pick an initial skeleton $\mathsf{initial}(M)$ using three distinct E-variables e_0, e_1, and e_2, and one T-variable a_0, as follows:

$$
\begin{array}{ll}
\mathsf{initial}(x) & = x^{:\mathsf{a}_0} \\
\mathsf{initial}(\lambda x.M) & = \begin{pmatrix} \text{let } Q = \mathsf{e}_0\, \mathsf{initial}(M) \\ \text{in } \lambda x.\, Q \end{pmatrix} \\
\mathsf{initial}(M \mathbin{@} N) & = \begin{pmatrix} \text{let } Q_1 = \mathsf{e}_1\, \mathsf{initial}(M) \\ \qquad Q_2 = \mathsf{e}_2\, \mathsf{initial}(N) \\ \text{in } Q_1{}^{:\mathsf{rtype}(Q_2) \to \mathsf{a}_0} \mathbin{@} Q_2 \end{pmatrix}
\end{array}
$$

We extend this definition to contexts, taking $\mathsf{initial}(\square) = \square$.

Example 4.1 (Initial skeleton). Let $M = (\lambda x.x \mathbin{@} x) @ y$, $Q = \mathsf{initial}(M)$:

$$
Q = \begin{array}{l|l}
(& \begin{array}{|l} \mathsf{e}_1\, (\lambda x.\, \mathsf{e}_0\, ((\mathsf{e}_1\, (x^{:\mathsf{a}_0}) : \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\, (x^{:\mathsf{a}_0}))) \end{array} \\
: & \begin{array}{|l} \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{a}_0) \end{array} \\
@ & \mathsf{e}_2\, (y^{:\mathsf{a}_0})
\end{array}
$$

We have $(M \triangleright Q) : \langle A \vdash \tau \rangle \ /\ \Delta$ where $A = (y : \mathsf{e}_2\, \mathsf{a}_0)$, $\tau = \mathsf{a}_0$ and $\Delta = (\mathsf{e}_1\, (\mathsf{e}_0\, \mathsf{e}_1\, \mathsf{a}_0 \mathbin{\dot\cap} \mathsf{e}_0\, \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{e}_0\, \mathsf{a}_0) \mathrel{\dot\leq} \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{a}_0) \mathbin{\dot\cap} (\mathsf{e}_1\, \mathsf{e}_0\, (\mathsf{e}_1\, \mathsf{a}_0 \mathrel{\dot\leq} \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{a}_0))$. ☐

We now define *pleasant* skeletons, which syntactically characterize skeletons produced by $\mathsf{initial}$.

Definition 4.2 (Pleasant skeleton). A skeleton Q is *pleasant* iff $Q = P$ for some P defined as follows:

$$
P ::= x^{:\mathsf{a}_0} \mid \lambda x.\, \mathsf{e}_0\, P \mid (\mathsf{e}_1\, P_1)^{:\mathsf{e}_2\, \tau \to \mathsf{a}_0} \mathbin{@} \mathsf{e}_2\, P_2
$$

Note that by convention 3.6, if $P = (\mathsf{e}_1\, P_1)^{:\mathsf{e}_2\, \tau \to \mathsf{a}_0} \mathbin{@} \mathsf{e}_2\, P_2$, then $\tau = \mathsf{rtype}(P_2)$. Note that this definition uses 4 specific, fixed concrete E- and T-variables (e_0, e_1, e_2, and a_0). ☐

All constraints of a pleasant skeleton are unsolved, and for all x, $\mathsf{tenv}(P)(x) = \vec{e}_1\, \mathsf{a}_0 \mathbin{\dot\cap} \cdots \mathbin{\dot\cap} \vec{e}_n\, \mathsf{a}_0$ for some $n \geq 0$.

Lemma 4.3 (Properties of $\mathsf{initial}$).

1. $\mathsf{initial}(M)$ *is a pleasant skeleton.*
2. *If* $P = \mathsf{initial}(M)$ *then* $\mathsf{term}(P) = M$.
3. *If* $M = \mathsf{term}(P)$ *then* $\mathsf{initial}(M) = P$. ☐

4.2 Readback

To show an exact correspondence with β-reduction, we define a *readback* function to reconstruct a pleasant skeleton from a typing and a constraint.

Definition 4.4 (Readback). Let $\mathsf{readback}$ be the least-defined function where $\mathsf{readback}(A, \tau, \Delta)$ is given by case analysis on τ and subsequently on Δ if $\tau = \mathsf{a}_0$, such that:

$$
\begin{array}{l}
\mathsf{readback}((x : \mathsf{a}_0), \mathsf{a}_0, \omega) = x^{:\mathsf{a}_0} \\
\mathsf{readback}(\mathsf{e}_0\, A, \mathsf{e}_0\, \tau_1 \to \mathsf{e}_0\, \tau_2, \mathsf{e}_0\, \Delta) = \lambda x.\, \mathsf{e}_0\, Q \\
\quad \text{if } A(x) = \omega \\
\quad \text{and } Q = \mathsf{readback}(A[x \mapsto \tau_1], \tau_2, \Delta) \\
\mathsf{readback}(\mathsf{e}_1\, A_1 \cap \mathsf{e}_2\, A_2, \mathsf{a}_0, \\
\qquad\qquad \mathsf{e}_1\, \Delta_1 \cap \mathsf{e}_2\, \Delta_2 \cap (\mathsf{e}_1\, \tau_1 \leq \mathsf{e}_2\, \tau_2 \to \mathsf{a}_0) \\
\quad = (\mathsf{e}_1\, Q_1)^{:\mathsf{e}_2\, \tau_2 \to \mathsf{a}_0} \mathbin{@} \mathsf{e}_2\, Q_2 \\
\quad \text{if } Q_1 = \mathsf{readback}(A_1, \tau_1, \Delta_1) \\
\quad \text{and } Q_2 = \mathsf{readback}(A_2, \tau_2, \Delta_2)
\end{array}
$$

For convenience, we let $\mathsf{readback}(Q) = \mathsf{readback}(\mathsf{tenv}(Q), \mathsf{rtype}(Q), \mathsf{unsolved}(\mathsf{constraint}(Q)))$. □

EXAMPLE 4.5 (READBACK). Let $A = (z : \mathsf{e}_2\, \mathsf{a}_0)$, $\tau = \mathsf{a}_0$ and:

$$
\Delta = \left| \begin{array}{l}
(\mathsf{e}_1\, (\mathsf{e}_0\, \mathsf{a}_0 \to \mathsf{e}_0\, \mathsf{a}_0) \leq \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{a}_0) \\
\cap\; (\mathsf{e}_1\, (\left| \begin{array}{l} (\mathsf{e}_0\, \mathsf{a}_0 \to \mathsf{e}_0\, \mathsf{a}_0) \to \mathsf{e}_0\, \mathsf{a}_0 \to \mathsf{e}_0\, \mathsf{a}_0 \\ \leq\; (\mathsf{e}_0\, \mathsf{a}_0 \to \mathsf{e}_0\, \mathsf{a}_0) \to \mathsf{e}_0\, \mathsf{a}_0 \to \mathsf{e}_0\, \mathsf{a}_0)) \end{array} \right.
\end{array} \right.
$$

We have:

$$
\mathsf{readback}(A, \tau, \Delta) = \left| \begin{array}{l}
(\mathsf{e}_1\, (\lambda x.\, \mathsf{e}_0\, (x^{:\mathsf{a}_0})) : \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{a}_0) \\
@\; \mathsf{e}_2\, (z^{:\mathsf{a}_0})
\end{array} \right.
$$

□

Lemma 4.6 (Readback builds pleasant skeletons). *If* $\mathsf{readback}(A, \tau, \Delta) = Q$, *then* $Q = P$ *for some* P. □

Lemma 4.7 (Readback is identity on pleasant skeletons). *If* $\mathsf{readback}(P) = P'$, *then* $P' = P$. □

Combining lemmas 4.3 and 4.7, we obtain this diagram:

$$
\begin{tikzcd}
M \arrow[r, bend left, "\mathsf{initial}"] & P \arrow[l, dashed, bend left, "\mathsf{term}"] \arrow[loop right, dashed, "\mathsf{readback}"]
\end{tikzcd}
$$

Lemma 4.8 (Readback preserves typings). *Suppose that* $\mathsf{readback}(A, \tau, \Delta) = P$ *and* $\mathsf{term}(P) = M$. *Then it holds that* $(M \rhd P) : \langle A \vdash \tau \rangle \,/\, \mathsf{unsolved}(\Delta)$. □

4.3 Unification Rules

From fig. 1, the metavariable $\dot{\Delta}$ ranges over *singular* constraints, those that contain exactly one type inequality; $\bar{\Delta}$ ranges over singular constraints that have an empty E-path. Each unification step solves one singular constraint.

Fig. 4 introduces rules $\mathsf{unify}\text{-}\beta$ and $\mathsf{unify}\text{-}@$, which are used to produce substitutions that solve singular constraints. Observe that these rules mention 4 specific, fixed concrete E- and T-variables (e_0, e_1, e_2, a_0), rather than arbitrary metavariables e and α. (Originally, we made the rules match arbitrary E-variables, but this generality was not actually used and the proof is simpler this way.) We now study each of these two rules in detail.

4.3.1 Rule $\mathsf{unify}\text{-}\beta$

This section shows that rule $\mathsf{unify}\text{-}\beta$ solves constraints in a way that corresponds exactly with β-reduction on λ-terms.

Rule $\mathsf{unify}\text{-}\beta$ matches on singular constraints of the form $\bar{\Delta} = \vec{e}\,(\mathsf{e}_1\, (\mathsf{e}_0\, \tau_0 \to \mathsf{e}_0\, \tau_1) \leq \mathsf{e}_2\, \tau_2 \to \mathsf{a}_0)$ corresponding to a β-redex $(\lambda x.M) \mathbin{@} N$ in the untyped λ-term. Informally, each portion of $\bar{\Delta}$ corresponds to a portion of the redex: τ_0 to the type of x in M, τ_1 to the result type of M, τ_2 to the result type of N, and a_0 to the result type of the whole redex.

EXAMPLE 4.9 (RULE $\mathsf{unify}\text{-}\beta$). Consider the following singular constraint $\bar{\Delta}$ which is part of Δ from example 4.1 (for memory, $M = (\lambda x.x \mathbin{@} x) \mathbin{@} y$):

$$\bar{\Delta} = (\mathsf{e}_1\, (\mathsf{e}_0\, \mathsf{e}_1\, \mathsf{a}_0 \cap \mathsf{e}_0\, \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{e}_0\, \mathsf{a}_0) \leq \mathsf{e}_2\, \mathsf{a}_0 \to \mathsf{a}_0)$$

This constraint corresponds to the β-redex in M. We have $\bar{\Delta} \xrightarrow{\mathsf{unify}\text{-}\beta} S$, where S is the following substitution:

$$
\left| \begin{array}{l}
\left| \begin{array}{l}
\mathsf{e}_2 := \mathsf{e}_1\, \mathsf{e}_0\, (\mathsf{e}_1 \boxdot \cap\, \mathsf{e}_2 \boxdot) \\
,\; \left| \begin{array}{l}
\mathsf{e}_1 := \mathsf{e}_1\, (\left| \begin{array}{l}
\mathsf{e}_0 := \mathsf{e}_0\, (\left| \begin{array}{l}
\mathsf{e}_1 := \mathsf{e}_1\, (\mathsf{a}_0 := \mathsf{a}_0 \,,\, \boxdot) \,,\, \boxdot \\
;\; \mathsf{e}_2 := \mathsf{e}_2\, (\mathsf{a}_0 := \mathsf{a}_0 \,,\, \boxdot) \,,\, \boxdot)
\end{array} \right. \\
,\; \boxdot)
\end{array} \right. \\
,\; \boxdot
\end{array} \right.
\end{array} \right. \\
;\; \mathsf{a}_0 := \mathsf{a}_0 \,,\, \mathsf{e}_1 := \mathsf{e}_0 := \boxdot \,,\, \boxdot \,,\, \boxdot
\end{array} \right.
$$

The first part of S (line 1, $\mathsf{e}_2 := \mathsf{e}_1\, \mathsf{e}_0\, (\mathsf{e}_1 \boxdot \cap\, \mathsf{e}_2 \boxdot)$) makes as many copies of argument y of the β-redex as there are occurences of the bound variable x. Each copy is put underneath a distinct sequence of E-variables (here e_1 and e_2); each sequence of E-variables corresponds to one occurence of x. The second part of the substitution S (lines 2-4) replaces the type variable associated with each occurence of the bound variable x with the result type of the corresponding copy of the argument y (this has no effect here, because y also has type a_0). The last part of S (line 5) simultaneously replaces the type variable that holds the result type of the application with the result type of the body of the λ-abstraction, and then erases E-variables e_1 and e_0. Only E-variable e_1 needs to be erased for the constraint to become solved, but we also erase e_0 in order to preserve definedness of $\mathsf{readback}$. □

Lemma 4.10. *If* $\bar{\Delta} \xrightarrow{\mathsf{unify}\text{-}\beta} S$, *then* $\mathsf{solved}([S]\, \bar{\Delta})$.

$$
\begin{tikzcd}
S & & \\
& \langle S, \bar{\Delta} \rangle \arrow[ul, "\pi_1"'] \arrow[dl, "\pi_2"] \arrow[r, dashed, "{[\cdot]\, \cdot;\, \mathsf{unsolved}}"] & \omega \\
\bar{\Delta} \arrow[uu, "\mathsf{unify}\text{-}\beta"] & &
\end{tikzcd}
$$

□

Proof. $\bar{\Delta} \xrightarrow{\mathsf{unify}\text{-}\beta} S$ implies

$$
\begin{array}{rcl}
\bar{\Delta} &=& \mathsf{e}_1\, (\mathsf{e}_0\, \tau_0 \to \mathsf{e}_0\, \tau_1) \leq \mathsf{e}_2\, \tau_2 \to \mathsf{a}_0 \\
\tau_0 &=& \vec{e}_1\, \mathsf{a}_0 \cap \cdots \cap \vec{e}_n\, \mathsf{a}_0 \\
S &=& \left| \begin{array}{l} (\mathsf{e}_2 := \mathsf{e}_1\, \mathsf{e}_0\, E \,,\, \mathsf{e}_1/\mathsf{e}_0/S') \\ ;\; (\mathsf{a}_0 := [S']\, \tau_1 \,,\, \mathsf{e}_1 := \mathsf{e}_0 := \boxdot) \end{array} \right. \\
\tau_0 &\xrightarrow{\mathsf{extract}}& E \quad \text{and} \quad \tau_0 \xrightarrow{:=\tau_2} S'.
\end{array}
$$

Let $E' = \vec{e}_1 \boxdot \cap \cdots \cap \vec{e}_n \boxdot$ and
$S'' = \vec{e}_1/(\mathsf{a}_0 := \tau_2); \cdots; \vec{e}_n/(\mathsf{a}_0 := \tau_2)$.
With these particular E' and S'', it is easy to see that $[E']\, \tau_2 = \vec{e}_1\, \tau_2 \cap \cdots \cap \vec{e}_n\, \tau_2 = [S'']\, \tau_0$ holds, provided that no $\vec{e}_i$ is a proper prefix of $\vec{e}_j$, with $i, j \in \{1, \cdots, n\}$. This is guaranteed by the hypothesis $\mathsf{constraint}(P) = \bar{\Delta} \cap \Delta'$, since for every i, $\vec{e}_i$ is the E-path of some skeleton variable in P. Because we have not imposed the equalities of sec. 3.2 on expansions and substitutions, it is not necessarily the case that $E = E'$ or $S' = S''$. However, since we have imposed the equalities of sec. 3.2 on types, all possible E and S' have the same effect when they are applied to a type: for all type τ, $[E]\, \tau = [E']\, \tau$ and $[S']\, \tau = [S'']\, \tau$. By definition 3.2, $[S]\, \bar{\Delta} = [S']\, \tau_0 \to [S]\, \tau_1 \leq [E]\, \tau_2 \to [S]\, \tau_1$. Since $[E]\, \tau_2 = [E']\, \tau_2 = [S'']\, \tau_0 = [S']\, \tau_0$, by definition of solvedness, $\mathsf{solved}([S]\, \bar{\Delta})$ holds. □

The next lemma shows that, given a pleasant skeleton P whose term is a β-redex, any substitution generated by $\mathsf{unify}\text{-}\beta$ applied to the constraint $\bar{\Delta}$ of P generates a substitution S that, when applied to P, (1) solves $\bar{\Delta}$ and only $\bar{\Delta}$, (2) preserves definedness of $\mathsf{readback}$, and (3) changes the term of the readback by contracting the β-redex considered.

$$\vec{e}(\mathsf{e}_1\ (\mathsf{e}_0\,\tau_0 \to \mathsf{e}_0\,\tau_1) \,\underline{\le}\, \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0) \xrightarrow{\text{unify-}\beta} \vec{e}/((\mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_0\,E,\ \mathsf{e}_1/\mathsf{e}_0/S); (\mathsf{a}_0 := [S]\,\tau_1,\ \mathsf{e}_1 := \mathsf{e}_0 := \boxdot))$$
$$\text{if } \tau_0 \xrightarrow{\text{extract}} E \text{ and } \tau_0 \xrightarrow{:=\tau_2} S$$

$$\vec{e}(\mathsf{e}_1\,\mathsf{a}_0 \,\underline{\le}\, \mathsf{e}_2\,\tau_1 \to \mathsf{a}_0) \xrightarrow{\text{unify-}@} \vec{e}/(\mathsf{e}_1 := (\mathsf{a}_0 := \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0,\ \mathsf{e}_1 := \mathsf{e}_1\,\mathsf{e}_1\,\boxdot,\ \mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_2\,\boxdot))$$

$$\begin{array}{lll|lll}
\alpha & \xrightarrow{\text{extract}} \boxdot & & \alpha & \xrightarrow{:=\tau_2} (\alpha := \tau_2) & \\
\tau_1 \cap \tau_2 & \xrightarrow{\text{extract}} E_1 \cap E_2 & \text{if } \tau_1 \xrightarrow{\text{extract}} E_1 \text{ and } \tau_2 \xrightarrow{\text{extract}} E_2 & \tau_1 \cap \tau_2 & \xrightarrow{:=\tau_2} S_1; S_2 & \text{if } \tau_1 \xrightarrow{:=\tau_2} S_1 \text{ and } \tau_2 \xrightarrow{:=\tau_2} S_2 \\
e\,\tau & \xrightarrow{\text{extract}} e\,E & \text{if } \tau \xrightarrow{\text{extract}} E & e\,\tau & \xrightarrow{:=\tau_2} e/S & \text{if } \tau \xrightarrow{:=\tau_2} S \\
\omega & \xrightarrow{\text{extract}} \omega & & \omega & \xrightarrow{:=\tau_2} \boxdot &
\end{array}$$

Figure 4: Rules for solving constraints.

Lemma 4.11. *Let $M = (\lambda x.M_1)\ @\ M_2$. If $P = \mathsf{initial}(M)$, $\mathsf{constraint}(P) = \bar\Delta \cap \Delta'$, and $\bar\Delta \xrightarrow{\text{unify-}\beta} S$, then there exists P' s.t. $\mathsf{constraint}(P') = [S]\,\Delta'$, $\mathsf{readback}([S]\,P) = P'$, and $\mathsf{term}(P') = M_1[x := M_2]$.* ☐

Proof. See appendix A.1. ☐

Lemma 4.12 (One step of unify-β corresponds to one step of β-reduction). *If $\mathsf{readback}(Q) = P = \mathsf{initial}(M)$, $M = \mathsf{term}(P)$, $\mathsf{constraint}(Q) \sqsupseteq \dot\Delta \xrightarrow{\text{unify-}\beta} S$, and $Q' = [S]\,Q$, then there exist M' and P' s.t. $\mathsf{readback}(Q') = P' = \mathsf{initial}(M')$, and $M \xrightarrow{\beta} M' = \mathsf{term}(P')$.*

$$\begin{tikzcd}
\dot\Delta \arrow[r, "\text{unify-}\beta"] & S & \\
Q \arrow[u, "\text{constraint};\, \sqsupseteq"] \arrow[d, "\text{readback}"'] & \langle S, Q\rangle \arrow[l, "\pi_2"] \arrow[u, "\pi_1"'] \arrow[r, "{[\cdot]\cdot}"'] & Q' \arrow[d, dashed, "\text{readback}"] \\
P \arrow[d, bend left, "\text{term}"] & & P' \arrow[d, dashed, bend left, "\text{term}"] \\
M \arrow[u, bend left, "\text{initial}"] \arrow[rr, dashed, "\beta"] & & M' \arrow[u, dashed, bend left, "\text{initial}"]
\end{tikzcd}$$ ☐

Proof. Let $\Delta = \mathsf{constraint}(Q)$ and $\Delta' = \mathsf{constraint}(P) = \mathsf{unsolved}(\Delta)$. We have $\Delta \sqsupseteq \dot\Delta = \vec{e}\,\bar\Delta \xrightarrow{\text{unify-}\beta} \vec{e}/S'$. Since unify-$\beta$ matches only unsolved constraints of a particular shape, we know that $\Delta' \sqsupseteq \dot\Delta$ and there exist D, C, M_1, x, M_2, M_3, P_3 s.t. $M_3 = (\lambda x.M_1)\ @\ M_2$, $P_3 = \mathsf{initial}(M_3)$, $P = D[P_3]$, $\vec{e} = \mathsf{E\text{-}path}(D)$, $\bar\Delta \sqsubseteq \mathsf{constraint}(P_3)$, and $M = C[M_3]$. Thus, $\bar\Delta \xrightarrow{\text{unify-}\beta} S'$, so by by lem. 4.11, there exists P_3' s.t. $\mathsf{readback}([S']\,P_3) = P_3'$ and $\mathsf{term}(P_3') = M_1[x := M_2]$. Consequently, $P' = D[P_3']$ and $M' = C[M_1[x := M_2]]$ satisfy $\mathsf{term}(P') = C[\mathsf{term}(P_3')] = M'$, $P' = \mathsf{initial}(M')$, and

$$\begin{aligned}
\mathsf{readback}([S]\,Q) &= \mathsf{readback}([\vec{e}/S']\,D[P_3]) \\
&= \mathsf{readback}(D[[S']\,P_3]) \\
&= \mathsf{readback}(D[P_3']) = D[P_3'] = P'. \quad \square.
\end{aligned}$$

Lemma 4.13 (One step of β-reduction corresponds to one step of unify-β). *If $\mathsf{readback}(Q) = P = \mathsf{initial}(M)$, $\mathsf{term}(P) = M \xrightarrow{\beta} M'$, $\mathsf{initial}(M') = P'$, and $\mathsf{term}(P') = M'$, then there exist $\dot\Delta$, S and Q' s.t. $\mathsf{constraint}(Q) \sqsupseteq \dot\Delta \xrightarrow{\text{unify-}\beta} S$, $[S]\,Q = Q'$, and $\mathsf{readback}(Q') = P'$.*

$$\begin{tikzcd}
\dot\Delta \arrow[r, dashed, "\text{unify-}\beta"] & S & \\
Q \arrow[u, dashed, "\text{constraint};\, \sqsupseteq"] \arrow[d, "\text{readback}"'] & \langle S, Q\rangle \arrow[l, dashed, "\pi_2"] \arrow[u, dashed, "\pi_1"'] \arrow[r, dashed, "{[\cdot]\cdot}"'] & Q' \arrow[d, dashed, "\text{readback}"] \\
P \arrow[d, bend left, "\text{term}"] & & P' \arrow[d, bend left, "\text{term}"] \\
M \arrow[u, bend left, "\text{initial}"] \arrow[rr, "\beta"] & & M' \arrow[u, bend left, "\text{initial}"]
\end{tikzcd}$$ ☐

Proof. Since $M \xrightarrow{\beta} M'$, there exist a context C and a redex $M_3 = (\lambda x.M_1)\ @\ M_2$ s.t. $M = C[M_3]$. Let $P_3 = \mathsf{initial}(M_3)$; by examination of P_3, we have $\mathsf{constraint}(P_3) \sqsupseteq \bar\Delta \xrightarrow{\text{unify-}\beta} S'$. Let $D = \mathsf{initial}(C)$ and $\vec{e} = \mathsf{E\text{-}path}(D)$; we have $P = D[P_3]$ and $\mathsf{constraint}(P) \sqsupseteq \dot\Delta = \vec{e}\,\bar\Delta \xrightarrow{\text{unify-}\beta} \vec{e}/S' = S$. Suppose $(N \triangleright Q) : \langle A \vdash \tau\rangle \,/\, \Delta$ and let $Q' = [S]\,Q$, $A' = [S]\,A$, $\tau' = [S]\,\tau$, and $\Delta' = [S]\,\Delta$.

By lem. 4.8, we have $(M \triangleright P) : \langle A \vdash \tau\rangle \,/\, \mathsf{unsolved}(\Delta)$. By lem. 4.11, there exists P_3' s.t. $\mathsf{readback}([S']\,P_3) = P_3'$ and $\mathsf{term}(P_3') = M_1[x := M_2] = M_3'$. Because S only operates inside the namespace $\vec{e} = \mathsf{E\text{-}path}(D)$, we have $[S]\,P = D[[S']\,P_3]$ and $(M \triangleright D[[S']\,P_3]) : \langle A' \vdash \tau'\rangle / [S]\,\mathsf{unsolved}(\Delta)$. By lem. 4.8, $\mathsf{readback}([S']\,P_3) = P_3'$ implies that $\mathsf{typing}([S']\,P_3) = \mathsf{typing}(P_3')$ and $\mathsf{unsolved}(\mathsf{constraint}([S']\,P_3)) = \mathsf{constraint}(P_3')$. Because $D = \mathsf{initial}(C)$, if instead of replacing the context hole of D with $[S']\,P_3$, we replace it with P_3' (a pleasant skeleton with the same typing as $[S']\,P_3$ and a subset of unsolved constraints), we obtain the derivation of the judgement $(M' \triangleright D[P_3']) : \langle A' \vdash \tau'\rangle \,/\, \mathsf{unsolved}(\Delta')$. We conclude by observing that $\mathsf{initial}(M') = \mathsf{initial}(C[M_3']) = \mathsf{initial}(C)[\mathsf{initial}(M_3')] = D[P_3'] = P' = \mathsf{readback}(P') = \mathsf{readback}(D[P_3']) = \mathsf{readback}(A', \tau', \mathsf{unsolved}(\Delta')) = \mathsf{readback}(Q')$. ☐

4.3.2 Rule unify-@

Rule unify-@ is designed to be used after rule unify-β has been used as much as possible and can not be used anymore. Because unify-β effectively simulates β-reduction, this means when unify-β is done, the term resulting from readback is a β-normal form. Because a β-normal form may have applications in it, there may still be unsolved constraints. Rule unify-@ applies to singular constraints of the form $\bar\Delta = \mathsf{e}_1\,\mathsf{a}_0 \,\underline{\le}\, \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0$. Such constraints are generated by terms that are chains of applications whose head is a variable, i.e., terms of the form $xM_1 \cdots M_n$. For $[S]\,\bar\Delta$ to be solved, S must replace $\mathsf{e}_1\,\mathsf{a}_0$ by an arrow type.

The substitution S produced by unify-@ is simple, but has been carefully designed. Part of the substitution produced by unify-@ is $S' = \mathsf{e}_1 := (\mathsf{a}_0 := \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0)$ which has the property that $[S']\,\bar\Delta$ is solved. However, S' erases e_1, which effectively merges the namespace inside e_1 with the parent namespace. Without care, this has the danger that it could cause formerly distinct variables which should remain distinct to become the same. This confusion of variables would break the principal typing property.

There are three variables that we have to be careful about that might live at the top level in the namespace of e_1 just before rule unify-@ is applied. The strategy we use for applying rules unify-β and unify-@, discussed in the next section, guarantees that e_0 does not occur at the top-level of the namespace inside e_1. We do not need to worry about the single T-variable a_0, because it is completely replaced by S. The rest of the substitution produced by unify-@ avoids confusing e_1 and e_2 (and also their nested namespaces) in the namespace of the outer e_1 with the same names in the parent namespace by replacing them by $\mathsf{e}_1\mathsf{e}_1$ and respectively $\mathsf{e}_1\mathsf{e}_2$. This is done at the same time as the outer e_1 is erased, thereby effectively preserving the original namespaces.

Note that if a constraint $\dot\Delta = \mathsf{e}_1\,(\mathsf{e}_1\,\mathsf{a}_0 \preceq \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0)$ exists in the namespace of the outer e_1 as it is erased, this constraint becomes $[S]\,\dot\Delta = \mathsf{e}_1\,\mathsf{e}_1\,\mathsf{a}_0 \preceq \mathsf{e}_1\,\mathsf{e}_2\,\tau_2 \to \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0$, to which unify-@ would not apply, and constraint solving would get stuck at some point. This issue is easily solved by applying unify-@ with a strategy that always selects the singular constraints that have the longer E-path first. Such a strategy is given in the next section.

4.4 Inference Algorithm

This section presents an algorithm producing a valid, solved skeleton for any β-normalizable term.

Let M be an arbitrarily chosen λ-term. If M has a β-normal form, then the procedure described below will terminate, otherwise it will go forever. The results of previous sections are used to design a strategy for applying rules unify-β and unify-@ that exactly follows leftmost/outermost β-reduction. This reduction strategy is known to terminate for arbitrary normalizing terms, so by following it we are able to infer typings for any term that has a β-normal form.

Definition 4.14 (Leftmost/outermost redex of an untyped term). Define metavariables and sets as follows:

$$\begin{array}{rcll} C^{\mathsf{lo1}} & \in & \mathsf{LO1\text{-}Context} & ::= \Box \mid C^{\mathsf{lo1}} \mathbin{@} M \mid M^{\mathsf{nf1}} \mathbin{@} C^{\mathsf{lo}} \\ C^{\mathsf{lo}} & \in & \mathsf{LO\text{-}Context} & ::= C^{\mathsf{lo1}} \mid \lambda x.C^{\mathsf{lo}} \\ M^{\mathsf{nf1}} & \in & \mathsf{NF1\text{-}Term} & ::= x \mid M^{\mathsf{nf1}} \mathbin{@} M^{\mathsf{nf}} \\ M^{\mathsf{nf}} & \in & \mathsf{NF\text{-}Term} & ::= M^{\mathsf{nf1}} \mid \lambda x.M^{\mathsf{nf}} \end{array}$$

For every term M, exactly one of the following conditions holds. (1) $M = M^{\mathsf{nf}}$, meaning M is a β-normal form. (2) $M = C^{\mathsf{lo}}[(\lambda x.M_1) \mathbin{@} M_2]$, in which case the occurrence of the subterm $(\lambda x.M_1) \mathbin{@} M_2$ in the hole of C^{lo} is the *leftmost/outermost redex of M*. □

In order to precisely define the complete strategy for type inference, we now define the notions of *leftmost/outermost constraint*, which we use for unify-β, and *rightmost/innermost constraint*, which we use for unify-@.

Definition 4.15 (Order on sequences of expansion variables). Let $\mathsf{e}_i < \mathsf{e}_j$ iff $i < j$. Thus, $<$ is a strict total order on expansion variables, and so is its lexicographic-extension order $<_{\mathsf{lex}}$ on sequences of finite length. □

Definition 4.16 (Leftmost/outermost constraint). If $\Delta = \vec{e}_1\,\bar\Delta_1 \Cap \cdots \Cap \vec{e}_n\,\bar\Delta_n$, then the *leftmost/outermost constraint* of Δ, written $\mathsf{LO}(\Delta)$, if it exists, is the singular constraint that has the least E-path, i.e., $\mathsf{LO}(\Delta) = \vec{e}_i\,\bar\Delta_i$ iff $1 \le i \le n$ and $\vec{e}_i <_{\mathsf{lex}} \vec{e}_j$ for any $j \in (\{1,\cdots,n\} \setminus \{i\})$. □

Definition 4.17 (Right/innermost constraint). If $\Delta = \vec{e}_1\,\bar\Delta_1 \Cap \cdots \Cap \vec{e}_n\,\bar\Delta_n$, then the *rightmost/innermost constraint* of Δ, written $\mathsf{RI}(\Delta)$, if it exists, is the singular constraint that has the greatest E-path, i.e., $\mathsf{RI}(\Delta) = \vec{e}_i\,\bar\Delta_i$ iff $1 \le i \le n$ and $\vec{e}_j <_{\mathsf{lex}} \vec{e}_i$ for any $j \in (\{1,\cdots,n\} \setminus \{i\})$. □

By design, the leftmost/outermost constraint of a pleasant skeleton P, namely $\mathsf{LO}(\mathsf{constraint}(P))$, corresponds exactly to the leftmost/outermost application in $\mathsf{term}(P)$. The leftmost/outermost constraint in $\mathsf{constraint}(P)$ that also matches rule unify-β corresponds exactly to the leftmost/outermost β-redex in $\mathsf{term}(P)$. Thus, the constraint solving strategy that always uses rule unify-β on the leftmost/outermost constraint matching unify-β corresponds exactly to leftmost/outermost β-reduction.

Let filter-$\beta(\Delta)$ contain all the singular constraints of Δ to which rule unify-β applies. That is, given $\Delta = \dot\Delta_1 \Cap \cdots \Cap \dot\Delta_n$, let $\text{filter-}\beta(\Delta) = \dot\Delta_{i_1} \Cap \cdots \Cap \dot\Delta_{i_p}$ where $j \in \{i_1,\cdots,i_p\} \subseteq \{1,\cdots,n\}$ iff there exists S s.t. $\dot\Delta_j \xrightarrow{\text{unify-}\beta} S$.

Lemma 4.18. *If* $\mathsf{readback}(Q) = P = \mathsf{initial}(M)$ *and* $M = C^{\mathsf{lo}}[(\lambda x.M_1) \mathbin{@} M_2]$, *then* $\dot\Delta = \mathsf{LO}(\text{filter-}\beta(\mathsf{constraint}(Q)))$ *is defined and* $\text{E-path}(\dot\Delta) = \text{E-path}(\mathsf{initial}(C^{\mathsf{lo}}))$.

$$\begin{tikzcd} Q \arrow[r, dashed, "\text{constraint; filter-}\beta\text{; LO}"'] \arrow[d, "\text{readback}"'] & \dot\Delta \arrow[r, dashed, "\text{E-path}"'] & \vec{e} \\ P \arrow[d, bend left, "\text{term}"] & & \\ M \arrow[u, bend left, "\text{initial}"] & & C^{\mathsf{lo}} \arrow[ll, "\cdot[(\lambda x.M_1) \mathbin{@} M_2]"'] \arrow[uu, dashed, "\text{initial; E-path}"'] \end{tikzcd}$$

□

Similarly, in $\mathsf{constraint}(P)$, the rightmost/innermost constraint corresponds to the rightmost/innermost application in $\mathsf{term}(\mathsf{readback}(Q))$. This precise correspondence breaks down once we start using rule unify-@, because readback is no longer defined after using unify-@. However, the intuition still explains how our strategy of using unify-@ works, because unify-@ does not rearrange namespaces. Appendix B presents a modified readback which is still defined after uses of unify-@. We do not use it here because its correspondence with pleasant skeletons is not very tight, and this would complicate the proofs.

Definition 4.19 (Unification algorithm). Given a term M, the strategy (which succeeds iff M has a β-normal form) for solving the constraint $\Delta = \mathsf{constraint}(\mathsf{initial}(M))$ is:

(0) Initial call: $\mathsf{Unify}(\Delta) \longrightarrow \mathsf{Unify}(\Delta, \boxdot)$
(1) $\mathsf{Unify}(\Delta, S) \longrightarrow \mathsf{Unify}(\mathsf{unsolved}([S']\,\Delta), S; S')$
    if $\mathsf{LO}(\text{filter-}\beta(\Delta)) \xrightarrow{\text{unify-}\beta} S'$
(2) $\mathsf{Unify}(\Delta, S) \longrightarrow \mathsf{Unify}(\mathsf{unsolved}([S']\,\Delta), S; S')$
    if $\mathsf{RI}(\Delta) \xrightarrow{\text{unify-}@} S'$ and case (1) does not apply
(3) Final call: $\mathsf{Unify}(\omega, S) \longrightarrow S$ □

Lemma 4.20 (Unify **succeeds for β-normal forms).** *If* $\mathsf{readback}(Q) = P$ *and* $\mathsf{term}(P) = M^{\mathsf{nf}}$, *there exists S s.t.* $\mathsf{Unify}(\mathsf{constraint}(Q)) \twoheadrightarrow S$ *and* $\mathsf{solved}([S]\,Q)$. □

Proof. Since $\mathsf{term}(P) = M^{\mathsf{nf}}$, $\Delta = \mathsf{constraint}(P)$ is of the form $\dot\Delta_1 \Cap \cdots \Cap \dot\Delta_n$ where each $\dot\Delta_i$ $(1 \le i \le n)$ can be solved by rule unify-@. Solving the rightmost/innermost constraint with unify-@ does not affect remaining unsolved constraints, so $\mathsf{Unify}(\Delta) \twoheadrightarrow S$, and $[S]\,Q$ is solved. □

Lemma 4.21. *If* $P = \mathsf{initial}(C[(\lambda x.M_1) \mathbin{@} M_2])$ *and* $\dot\Delta = \text{E-path}(\mathsf{initial}(C))\,\bar\Delta \sqsubseteq \mathsf{constraint}(P)$ *then* $\dot\Delta \xrightarrow{\text{unify-}\beta} S$ *and* $\mathsf{term}(\mathsf{readback}([S]\,P)) = C[M_1[x := M_2]]$. □

Proof. By induction on C, using lem. 4.10 and 4.11. □

Lemma 4.22 (Unify **succeeds for β-normalizing terms).** *If* $\mathsf{readback}(Q) = P$ *and* $M = \mathsf{term}(P) \xrightarrow{\beta}\!\!\!\!\rightarrow M^{\mathsf{nf}}$, *then*

$\mathsf{Unify}(\mathsf{constraint}(Q)) \longrightarrow\!\!\!\!\rightarrow S$ *and* $\mathsf{solved}([S]\,Q)$.

$$\begin{tikzcd}
\Delta \arrow[r, dashed, "\mathsf{Unify}"] & S & \\
Q \arrow[u, dashed, "\mathsf{constraint}"] \arrow[d, "\mathsf{readback}"'] & \langle S, Q\rangle \arrow[l, dashed, "\pi_2"'] \arrow[u, dashed, "\pi_1"'] \arrow[r, dashed, "{[\cdot]\,\cdot}"] & Q' \arrow[d, dashed, "\mathsf{constraint}"] \\
P \arrow[d, bend left, "\mathsf{term}"] & & \omega \\
M \arrow[u, bend left, "\mathsf{initial}"] \arrow[rr, two heads, "\beta"] & & M^{\mathsf{nf}}
\end{tikzcd}$$ □

Proof. If $M = \mathsf{term}(P) \xrightarrow{\beta}\!\!\!\!\rightarrow M^{\mathsf{nf}}$, then it does so in n steps following a leftmost/outermost strategy. The proof is by induction on n. Case $n = 0$: direct by lem. 4.20. Case $n = m+1$: $M = C^{\mathsf{lo}}[(\lambda x.M_1) @ M_2] \xrightarrow{\beta} C^{\mathsf{lo}}[M_1[x := M_2]] = M'$ and $M' \xrightarrow{\beta}\!\!\!\!\rightarrow M^{\mathsf{nf}}$ in m steps. Let $\Delta = \mathsf{constraint}(P) = \mathsf{unsolved}(\mathsf{constraint}(Q))$. By lem. 4.18, there exists $\dot{\Delta} = \mathsf{LO}(\mathsf{filter}\text{-}\beta(\Delta)) \sqsubseteq \Delta$ s.t. $\mathsf{E\text{-}path}(\dot{\Delta}) = \mathsf{E\text{-}path}(\mathsf{initial}(C^{\mathsf{lo}}))$. By lem. 4.21, $\dot{\Delta} \xrightarrow{\mathsf{unify}\text{-}\beta} S_1$ and there exists P' s.t. $P' = \mathsf{readback}([S_1]\,Q)$ and $\mathsf{term}(P') = M'$. By the induction hypothesis on $[S_1]\,Q$,, P', and M', there exists a substitution S_2 s.t. $\mathsf{Unify}(\mathsf{constraint}([S_1]\,Q)) \longrightarrow\!\!\!\!\rightarrow S_2$ and $\mathsf{solved}([S_2]\,[S_1]\,Q)$. Finally, by lem. 3.3, $S = S_1; S_2$ satisfies $\mathsf{Unify}(\mathsf{constraint}(Q)) \longrightarrow\!\!\!\!\rightarrow S$ and $\mathsf{solved}([S]\,Q)$. □

Definition 4.23 (Type inference algorithm). The overall algorithm is:

$$\begin{aligned}
&\mathsf{Infer}(M) \longrightarrow \langle P, S\rangle \\
&\quad \text{if } P = \mathsf{initial}(M) \\
&\quad \text{and } \mathsf{Unify}(\mathsf{constraint}(P)) \longrightarrow\!\!\!\!\rightarrow S
\end{aligned}$$ □

If $\mathsf{Infer}(M) \longrightarrow \langle P, S\rangle$, a solved skeleton (traditional typing derivation) for M is obtained by computing $[S]\,P$. If only a typing (type environment, result type) is desired, it is obtained by computing $\langle [S]\,\mathsf{tenv}(P) \vdash [S]\,\mathsf{rtype}(P)\rangle$.

Example 4.24. Consider $M = (\lambda x.x @ x)@(\lambda z.z @ y)$. Appendix C.2 shows each type inference step for M, produced by the interactive version of our implementation, available at `http://www.macs.hw.ac.uk/ultra/compositional-analysis/system-e/`. The result shows that term M can be assigned the typing $\langle (y : (\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \sqcap \mathsf{e}_2\,\mathsf{a}_0) \vdash \mathsf{a}_0\rangle$ w.r.t. some solved constraint Δ, which expresses the fact that the β-normal form of M is $y @ y$. □

Theorem 4.25 (Infer succeeds for β-normalizing terms). *If $M \xrightarrow{\beta}\!\!\!\!\rightarrow M^{\mathsf{nf}}$, then $\mathsf{Infer}(M) \longrightarrow \langle P, S\rangle$ where $Q = [S]\,P$ is solved and $\mathsf{term}(Q) = M$.* □

Proof. Let $P = \mathsf{initial}(M)$. By lem. 4.7, $\mathsf{readback}(P) = P$. By lem. 4.22, there exists S s.t. $\mathsf{Unify}(\mathsf{constraint}(P)) \longrightarrow\!\!\!\!\rightarrow S$ and $\mathsf{solved}([S]\,P)$. By def. 3.2 and lem. 4.3, $\mathsf{term}([S]\,P) = \mathsf{term}(P) = M$. □

5. REFERENCES

[1] A. Banerjee. A modular, polyvariant, and type-based closure analysis. In *Proc. 1997 Int'l Conf. Functional Programming*. ACM Press, 1997.

[2] H. Barendregt, M. Coppo, M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symbolic Logic*, 48(4), 1983.

[3] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.

[4] G. Boudol, P. Zimmer. On type inference in the intersection type discipline. Draft., 2004.

[5] S. Carlier. Polar type inference with intersection types and ω. In *Proceedings of the 2nd Workshop on Intersection Types and Related Systems*, 2002. The ITRS '02 proceedings appears as vol. 70, issue 1 of *Elec. Notes in Theoret. Comp. Sci.*

[6] S. Carlier, J. Polakow, J. B. Wells, A. J. Kfoury. System E: Expansion variables for flexible typing with linear and non-linear types and intersection types. In *Programming Languages & Systems, 13th European Symp. Programming*, LNCS. Springer-Verlag, 2004.

[7] M. Coppo, F. Damiani, P. Giannini. Strictness, totality, and non-standard type inference. *Theoret. Comput. Sci.*, 272(1-2), 2002.

[8] M. Coppo, M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Logic*, 21(4), 1980.

[9] M. Coppo, M. Dezani-Ciancaglini, B. Venneri. Principal type schemes and λ-calculus semantics. In Hindley and Seldin [17].

[10] F. Damiani. A conjunctive type system for useless-code elimination. *Mathematical Structures in Computer Science*, 13, 2003.

[11] F. Damiani. Rank 2 intersection types for local definitions and conditional expressions. *ACM Trans. on Prog. Langs. & Systs.*, 25(4), 2003.

[12] F. Damiani. Rank 2 Intersection Types for Modules. In *PPDP'03*. ACM, 2003.

[13] F. Damiani, P. Giannini. Automatic useless-code detection and elimination for HOT functional programs. *J. Funct. Programming*, 2000.

[14] J.-Y. Girard. Linear logic: its syntax and semantics. In J.-Y. Girard, Y. Lafont, L. Regnier, eds., *Advances in Linear Logic, Proceedings of the 1993 Workshop on Linear Logic*, London Mathematical Society Lecture Note Series. Cambridge University Press, 1995.

[15] C. Haack, J. B. Wells. Type error slicing in implicitly typed, higher-order languages. In *Programming Languages & Systems, 12th European Symp. Programming*, vol. 2618 of *LNCS*. Springer-Verlag, 2003. Superseded by [16].

[16] C. Haack, J. B. Wells. Type error slicing in implicitly typed, higher-order languages. *Sci. Comput. Programming*, 50, 2004. To appear. Supersedes [15].

[17] J. R. Hindley, J. P. Seldin, eds. *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.

[18] T. Jensen. Inference of polymorphic and conditional strictness properties. In *Conf. Rec. POPL '98: 25th ACM Symp. Princ. of Prog. Langs.*, 1998.

[19] T. Jim. What are principal typings and what are they good for? In *Conf. Rec. POPL '96: 23rd ACM Symp. Princ. of Prog. Langs.*, 1996.

[20] A. J. Kfoury, J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Conf. Rec. POPL '99: 26th ACM Symp. Princ. of Prog. Langs.*, 1999. Superseded by [22].

[21] A. J. Kfoury, J. B. Wells. Principality and type inference for intersection types using expansion variables. Supersedes [20], 2003.

[22] A. J. Kfoury, J. B. Wells. Principality and type inference for intersection types using expansion variables. *Theoret. Comput. Sci.*, 311(1–3), 2004. Supersedes [20]. For omitted proofs, see the longer report [21].

[23] D. Leivant. Polymorphic type inference. In *Conf. Rec. 10th Ann. ACM Symp. Princ. of Prog. Langs.*, 1983.

[24] E. K. G. Lopez-Escobar. Proof-functional connectives. In C. Di Prisco, ed., *Methods of Mathematical Logic, Proceedings of the 6th Latin-American Symposium on Mathematical Logic, Caracas 1983*, vol. 1130 of *Lecture Notes in Mathematics*. Springer-Verlag, 1985.

[25] H. G. Mairson. Deciding ML typability is complete for deterministic exponential time. In *Conf. Rec. 17th Ann. ACM Symp. Princ. of Prog. Langs.*, 1990.

[26] H. G. Mairson. From Hilbert spaces to Dilbert spaces: Context semantics made simple. In *22nd Conference on Foundations of Software Technology and Theoretical Computer Science*, 2002.

[27] R. Milner. A theory of type polymorphism in programming. *J. Comput. System Sci.*, 17, 1978.

[28] P. Møller Neergaard, H. G. Mairson. Rank bounded intersection: Types, potency, and idempotency. Submitted, 2003.

[29] G. Pottinger. A type assignment for the strongly normalizable λ-terms. In Hindley and Seldin [17].

[30] L. Regnier. *Lambda calcul et réseaux*. PhD thesis, University Paris 7, 1992.

[31] S. Ronchi Della Rocca. Principal type schemes and unification for intersection type discipline. *Theoret. Comput. Sci.*, 59(1–2), 1988.

[32] S. Ronchi Della Rocca, B. Venneri. Principal type schemes for an extended type theory. *Theoret. Comput. Sci.*, 28(1–2), 1984.

[33] É. Sayag, M. Mauny. A new presentation of the intersection type discipline through principal typings of normal forms. Technical Report RR-2998, INRIA, 1996.

[34] P. Urzyczyn. Type reconstruction in $\mathbf{F}_\omega$. *Math. Structures Comput. Sci.*, 7(4), 1997.

[35] S. J. van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems.* PhD thesis, Catholic University of Nijmegen, 1993.

[36] S. J. van Bakel. Intersection type assignment systems. *Theoret. Comput. Sci.*, 151(2), 1995.

[37] J. B. Wells. The essence of principal typings. In *Proc. 29th Int'l Coll. Automata, Languages, and Programming*, vol. 2380 of *LNCS*. Springer-Verlag, 2002.

APPENDIX

A. PROOFS

A.1 Lemma 4.11

The following two lemmas are used as part of the proof of lem. 4.11.

Lemma A.1. *If* $\mathsf{e}_1\,\tau_1 \sqcap \mathsf{e}_2\,\tau_2 \xrightarrow{\mathsf{extract}} E$, *then there exist* E_1 *and* E_2 *s.t.* $\tau_1 \xrightarrow{\mathsf{extract}} E_1$, $\tau_2 \xrightarrow{\mathsf{extract}} E_2$, *and for all type or constraint* T, $[E]\,T = \mathsf{e}_1\,[E_1]\,T \sqcap \mathsf{e}_2\,[E_2]\,T$. ☐

Lemma A.2. *If* $\mathsf{e}_1\,\tau_1 \sqcap \mathsf{e}_2\,\tau_2 \xrightarrow{:=\tau} S$, *then there exist* S_1 *and* S_2 *s.t.* $\tau_1 \xrightarrow{:=\tau} S_1$, $\tau_2 \xrightarrow{:=\tau} S_2$, *and for all type or constraint* T *and* $i \in \{1,2\}$, $[S]\,\mathsf{e}_i\,T = \mathsf{e}_i\,[S_i]\,T$. ☐

Lemma lem. 4.11 is:
Assumptions:

- $M = (\lambda x.M_1) \mathbin{@} M_2$,
- $P = \mathsf{initial}(M)$,
- $\mathsf{constraint}(P) = \bar\Delta \sqcap \Delta'$,
- $\bar\Delta \xrightarrow{\mathsf{unify}\text{-}\beta} S$.

We need to prove that there exists P' s.t.

- $\mathsf{constraint}(P') = [S]\,\Delta'$,
- $\mathsf{readback}([S]\,P) = P'$, and
- $\mathsf{term}(P') = M_1[x := M_2]$.

Proof. By induction on M_1. By definition of initial, $P = (\mathsf{e}_1\,\lambda x.\,\mathsf{e}_0\,P_1)^{:\mathsf{e}_2\,\tau_2 \to \mathsf{a}_0} \mathbin{@} \mathsf{e}_2\,P_2$ and $(M \triangleright P) : \langle A \vdash \mathsf{a}_0\rangle \;/\; \Delta$, where

- $P_1 = \mathsf{initial}(M_1)$ and $(M_1 \triangleright P_1) : \langle A_1 \vdash \tau_1\rangle \;/\; \Delta_1$,
- $P_2 = \mathsf{initial}(M_2)$ and $(M_2 \triangleright P_2) : \langle A_2 \vdash \tau_2\rangle \;/\; \Delta_2$,
- $A = \mathsf{e}_1\,\mathsf{e}_0\,A_1[x \mapsto \omega] \sqcap \mathsf{e}_2\,A_2$,
- $\Delta = \mathsf{e}_1\,\mathsf{e}_0\,\Delta_1 \sqcap \bar\Delta \sqcap \mathsf{e}_2\,\Delta_2$,
- $\tau_0 = A_1(x) = \vec{e}_1\,\mathsf{a}_0 \sqcap \cdots \sqcap \vec{e}_n\,\mathsf{a}_0$, and
- $\bar\Delta = \mathsf{e}_1\,(\mathsf{e}_0\,\tau_0 \to \mathsf{e}_0\,\tau_1) \le \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0$.

Because $\bar\Delta \xrightarrow{\mathsf{unify}\text{-}\beta} S$, there exist E and S' s.t. $\tau_0 \xrightarrow{\mathsf{extract}} E$, $\tau_0 \xrightarrow{:=\tau_2} S'$, and $S = (\mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_0\,E,\ \mathsf{e}_1/\mathsf{e}_0/S')$; $(\mathsf{a}_0 := [S']\,\tau_1,\ \mathsf{e}_1 := \mathsf{e}_0 := \boxdot)$.

There are four cases to consider:

- Case $M_1 = y$ and $x \neq y$. $P_1 = \mathsf{initial}(M_1) = y^{:\mathsf{a}_0}$.
  $(M_1 \triangleright P_1) : \langle (y : \mathsf{a}_0) \vdash \mathsf{a}_0\rangle \;/\; \omega$.
  $\tau_0 = \omega$, $n = 0$, $E = \omega$, $S' = \boxdot$.
  $(M \triangleright [S]\,P) : \langle A_1[x \mapsto \omega] \sqcap \omega \vdash \tau_1\rangle \;/\; \omega \sqcap [S]\,\bar\Delta \sqcap \omega$.
  Let $P' = P_1$. $\mathsf{term}(P') = y = y[x := M_2] = M_1[x := M_2]$. $\mathsf{constraint}(P') = \omega = [S']\,\omega \sqcap [\omega]\,\Delta_2 = [S]\,\Delta'$. By lem. 4.7, $\mathsf{readback}([S]\,P) = \mathsf{readback}(A_1, \tau_1, \omega) = P_1 = P'$.

- Case $M_1 = x$. $P_1 = \mathsf{initial}(M_1) = x^{:\mathsf{a}_0}$.
  $(M_1 \triangleright P_1) : \langle (x : \mathsf{a}_0) \vdash \mathsf{a}_0\rangle \;/\; \omega$.
  $\tau_0 = \mathsf{a}_0$, $n = 1$, $E = \boxdot$, $S' = \mathsf{a}_0 := \tau_2$.
  $(M \triangleright [S]\,P) : \langle A_2 \vdash \tau_2\rangle \;/\; \omega \sqcap [S]\,\bar\Delta \sqcap \Delta_2$
  Let $P' = P_2$. $\mathsf{term}(P') = M_2 = x[x := M_2] = M_1[x := M_2]$. $\mathsf{constraint}(P') = \Delta_2 = [S']\,\omega \sqcap [E]\,\Delta_2 = [S]\,\Delta'$. By lem. 4.7, $\mathsf{readback}([S]\,P) = \mathsf{readback}(A_2, \tau_2, \Delta_2) = P_2 = P'$.

- Case $M_1 = \lambda y.M_3$ and $x \neq y$. Suppose w/o.l.o.g. that y does not occur free in M_2; hence, $A_2(y) = \omega$.
  Let $P_3 = \mathsf{initial}(M_3)$, and suppose that P_3 derives $(M_3 \triangleright P_3) : \langle A_3 \vdash \tau_3\rangle \;/\; \Delta_3$. We have $P_1 = \lambda y.\,\mathsf{e}_0\,P_3$, $A_1 = (\mathsf{e}_0\,A_3)[y \mapsto \omega]$, $\tau_1 = \mathsf{e}_0\,A_3(y) \to \mathsf{e}_0\,\tau_3$, and $\Delta_1 = \mathsf{e}_0\,\Delta_3$.
  Let $\tau_0' = A_3(x)$. $\tau_0 = \mathsf{e}_0\,\tau_0'$. There exist E_4 and S_4' s.t. $\tau_0' \xrightarrow{\mathsf{extract}} E_4$ and $\tau_0' \xrightarrow{:=\tau_2} S_4'$, determined as follows: If $\tau_0 = \omega$, then $E = E_4 = \omega$ and $S = S_4' = \boxdot$. If $\tau_0' \neq \omega$, then $E = \mathsf{e}_0\,E_4$, and $S = \mathsf{e}_0/S_4'$. Note that for all type or constraint T, $[E]\,T = [\mathsf{e}_0\,E_4]\,T$, and $[S']\,\mathsf{e}_0\,T = \mathsf{e}_0\,[S_4']\,T$.
  $[S]\,P$ derives the following judgement:
  $(M \triangleright [S]\,P) : \langle [S]\,A \vdash [S]\,\mathsf{a}_0\rangle \;/\; [S]\,\Delta$, where
  $[S]\,A = \mathsf{e}_0\,([S_4']\,A_3)[y \mapsto \omega][x \mapsto \omega] \sqcap \mathsf{e}_0\,[E_4]\,A_2$,
  $[S]\,\mathsf{a}_0 = [S']\,\tau_1 = \mathsf{e}_0\,([S_4']\,A_3(y)) \to \mathsf{e}_0\,[S_4']\,\tau_3$,
  $[S]\,\Delta = \mathsf{e}_0\,[S_4']\,\Delta_3 \sqcap [S]\,\bar\Delta \sqcap \mathsf{e}_0\,[E_4]\,\Delta_2$

  Let $M_4 = (\lambda x.M_3) \mathbin{@} M_2$ and $P_4 = \mathsf{initial}(M_4)$.
  P_4 derives the following judgement:
  $(M_4 \triangleright P_4) : \langle A_4 \vdash \mathsf{a}_0\rangle \;/\; \Delta_4$, where
  $A_4 = \mathsf{e}_1\,\mathsf{e}_0\,A_3[x \mapsto \omega] \sqcap \mathsf{e}_2\,A_2$,
  $\Delta_4 = \bar\Delta_4 \sqcap \Delta_4'$
  $\Delta_4' = \mathsf{e}_1\,\mathsf{e}_0\,\Delta_3 \sqcap \mathsf{e}_2\,\Delta_2$, and
  $\bar\Delta_4 = \mathsf{e}_1\,(\mathsf{e}_0\,\tau_0' \to \mathsf{e}_0\,\tau_3) \le \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0$.
  Let $S_4 = \left|\begin{array}{l}(\mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_0\,E_4\,,\ \mathsf{e}_1/\mathsf{e}_0/S_4') \\ ;\ (\mathsf{a}_0 := [S_4']\,\tau_3\,,\ \mathsf{e}_1 := \mathsf{e}_0 := \boxdot).\end{array}\right.$
  Since $\tau_0' \xrightarrow{\mathsf{extract}} E_4$ and $\tau_0' \xrightarrow{:=\tau_2} S_4'$, $\bar\Delta_4 \xrightarrow{\mathsf{unify}\text{-}\beta} S_4$. By I.H., there exists P_4' s.t.
  $\mathsf{readback}([S_4]\,P_4) = P_4'$,
  $\mathsf{term}(P_4') = M_3[x := M_2]$, and
  $\mathsf{constraint}(P_4') = [S_4]\,\Delta_4'$.
  P_4' derives the following judgement:
  $(P_4' \triangleright M_3[x := M_2]) : \langle [S_4]\,A_4 \vdash [S_4]\,\mathsf{a}_0\rangle \;/\; [S_4]\,\Delta_4'$.

  Let $P' = \lambda y.\,\mathsf{e}_0\,P_4'$. P' derives this judgement:
  $(M' \triangleright P') : \langle A' \vdash \tau'\rangle \;/\; \Delta''$ where

$$
\begin{aligned}
M' &= \lambda y.(M_3[x := M_2]) \\
&= (\lambda y.M_3)[x := M_2] \\
&= M_1[x := M_2] \\
A' &= (\mathsf{e}_0\,[S_4]\,P_4)[y \mapsto \omega] \\
&= (\mathsf{e}_0\,(([S_4']\,A_3)[x \mapsto \omega] \sqcap [E_4]\,A_2))[y \mapsto \omega] \\
&= \mathsf{e}_0\,((([S_4']\,A_3)[x \mapsto \omega] \sqcap [E_4]\,A_2)[y \mapsto \omega]) \\
&= \mathsf{e}_0\,(([S_4']\,A_3)[x \mapsto \omega][y \mapsto \omega] \sqcap ([E_4]\,A_2)[y \mapsto \omega]) \\
&= \mathsf{e}_0\,(([S_4']\,A_3)[y \mapsto \omega][x \mapsto \omega] \sqcap [E_4]\,(A_2[y \mapsto \omega])) \\
&= \mathsf{e}_0\,(([S_4']\,A_3)[y \mapsto \omega][x \mapsto \omega] \sqcap [E_4]\,A_2) \\
&= [S]\,A \\
\tau' &= \mathsf{e}_0\,A_4'(y) \to \mathsf{e}_0\,[S_4]\,\mathsf{a}_0 \\
&= \mathsf{e}_0\,A_4'(y) \to \mathsf{e}_0\,[S_4']\,\tau_3 \\
&= \mathsf{e}_0\,(([S_4']\,A_3)[x \mapsto \omega] \sqcap [E_4]\,A_2)(y) \to \mathsf{e}_0\,[S_4']\,\tau_3 \\
&= \mathsf{e}_0\,(([S_4']\,A_3)[x \mapsto \omega])(y) \to \mathsf{e}_0\,[S_4']\,\tau_3 \\
&= \mathsf{e}_0\,([S_4']\,A_3)(y) \to \mathsf{e}_0\,[S_4']\,\tau_3 \\
&= \mathsf{e}_0\,[S_4']\,A_3(y) \to \mathsf{e}_0\,[S_4']\,\tau_3 \\
&= [S]\,\mathsf{a}_0
\end{aligned}
$$

$\Delta'' = \mathsf{e}_0\,[S_4]\,\Delta'_4$
$\quad = \mathsf{e}_0\,([S'_4]\,\Delta_3 \mathbin{\dot\cap} [E_4]\,\Delta_2)$
$\quad = [S]\,\Delta'$

Since P' is a pleasant skeleton, $\mathsf{unsolved}(\Delta'') = \Delta''$. By lem. 4.10, $\mathsf{solved}([S]\,\bar{\Delta})$ holds. Using lem. 4.7,

$\mathsf{readback}([S]\,P)$
$\quad = \mathsf{readback}([S]\,A, [S]\,\mathsf{a}_0, \mathsf{unsolved}([S]\,\Delta))$
$\quad = \mathsf{readback}([S]\,A, [S]\,\mathsf{a}_0, \mathsf{unsolved}([S]\,(\bar{\Delta} \mathbin{\dot\cap} \Delta')))$
$\quad = \mathsf{readback}([S]\,A, [S]\,\mathsf{a}_0, \mathsf{unsolved}([S]\,\bar{\Delta} \mathbin{\dot\cap} [S]\,\Delta'))$
$\quad = \mathsf{readback}([S]\,A, [S]\,\mathsf{a}_0, \mathsf{unsolved}([S]\,\Delta'))$
$\quad = \mathsf{readback}([S]\,A, [S]\,\mathsf{a}_0, \mathsf{unsolved}(\Delta''))$
$\quad = \mathsf{readback}(A', \tau', \Delta'')$
$\quad = P'$.

- Case $M_1 = M_3 \mathbin{@} M_4$. Let $P_3 = \mathsf{initial} M_3$ and $P_4 = \mathsf{initial} M_4$. Assume that P_3 derives $(M_3 \triangleright P_3) : \langle A_3 \vdash \tau_3 \rangle \;/\; \Delta_3$ and P_4 derives $(M_4 \triangleright P_4) : \langle A_4 \vdash \tau_4 \rangle \;/\; \Delta_4$. $P_1 = (\mathsf{e}_1\,P_3)^{:\tau_4 \to \mathsf{a}_0} \mathbin{@} \mathsf{e}_2\,P_4$. $A_1 = \mathsf{e}_1\,A_3 \mathbin{\dot\cap} \mathsf{e}_2\,A_4$. $\tau_1 = \mathsf{a}_0$. $\Delta_1 = \mathsf{e}_1\,\Delta_3 \mathbin{\dot\cap} (\mathsf{e}_1\,\tau_3 \leq \mathsf{e}_3\,\tau_4 \to \mathsf{a}_0) \mathbin{\dot\cap} \mathsf{e}_2\,\Delta_4$.
Let $\tau'_3 = A_3(x)$ and $\tau'_4 = A_4(x)$. $\tau_0 = \mathsf{e}_1\,\tau'_3 \mathbin{\dot\cap} \mathsf{e}_2\,\tau'_4$.
By lem. A.1 and lem. A.2, there exist E_3, S'_3, E_4, E'_4 s.t. $\tau'_3 \xrightarrow{\mathsf{extract}} E_3$, $\tau'_3 \xrightarrow{:=\tau_2} S'_3$, $\tau'_4 \xrightarrow{\mathsf{extract}} E_4$, $\tau'_4 \xrightarrow{:=\tau_2} S'_4$, and for all type or constraint T, $[E_1]\,T = \mathsf{e}_1\,[E_3]\,T \mathbin{\dot\cap} \mathsf{e}_2\,[E_4]\,T$, $[S]\,\mathsf{e}_1\,T = \mathsf{e}_1\,[S'_3]\,T$, and $[S]\,\mathsf{e}_2\,T = \mathsf{e}_2\,[S'_4]\,T$.
$[S]\,P$ derives the following judgement:
$(M \triangleright [S]\,P) : \langle [S]\,A \vdash [S]\,\mathsf{a}_0 \rangle \;/\; [S]\,\Delta$, where
$$[S]\,A = \begin{array}{r|l} & (\mathsf{e}_1\,[S'_3]\,A_3 \mathbin{\dot\cap} \mathsf{e}_2\,[S'_4]\,A_4)[x \mapsto \omega] \\ \dot\cap & \mathsf{e}_1\,[E_3]\,A_2 \mathbin{\dot\cap} \mathsf{e}_2\,[E_4]\,A_2 \end{array}$$
$[S]\,\mathsf{a}_0 = [S']\,\tau_1 = [S'_3]\,\mathsf{a}_0 \mathbin{\dot\cap} [S'_4]\,\mathsf{a}_0$,
$$[S]\,\Delta = \begin{array}{r|l} & \mathsf{e}_1\,[S'_3]\,\Delta_3 \mathbin{\dot\cap} \mathsf{e}_2\,[S'_4]\,\Delta_4 \mathbin{\dot\cap} [S]\,\bar{\Delta} \\ \dot\cap & \mathsf{e}_1\,[E_3]\,\Delta_2 \mathbin{\dot\cap} \mathsf{e}_2\,[E_4]\,\Delta_2 \end{array}$$

Let $M_5 = (\lambda x.M_3) \mathbin{@} M_2$ and $P_5 = \mathsf{initial}(M_5)$.
P_5 derives the following judgement:
$(M_5 \triangleright P_5) : \langle A_5 \vdash \mathsf{a}_0 \rangle \;/\; \Delta_5$, where
$\quad A_5 = \mathsf{e}_1\,\mathsf{e}_0\,A_3[x \mapsto \omega] \mathbin{\dot\cap} \mathsf{e}_2\,A_2$,
$\quad \Delta_5 = \bar{\Delta}_5 \mathbin{\dot\cap} \Delta'_5$
$\quad \Delta'_5 = \mathsf{e}_1\,\mathsf{e}_0\,\Delta_3 \mathbin{\dot\cap} \mathsf{e}_2\,\Delta_2$, and
$\quad \bar{\Delta}_5 = \mathsf{e}_1\,(\mathsf{e}_0\,\tau'_3 \to \mathsf{e}_0\,\tau_3) \leq \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0$.
$$\text{Let } S_5 = \begin{array}{r|l} & (\mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_0\,E_3\,,\, \mathsf{e}_1/\mathsf{e}_0/S'_3) \\ ; & (\mathsf{a}_0 := [S'_3]\,\tau_3\,,\, \mathsf{e}_1 := \mathsf{e}_0 := \boxdot). \end{array}$$
Since $\tau'_3 \xrightarrow{\mathsf{extract}} E_3$ and $\tau'_3 \xrightarrow{:=\tau_2} S'_3$, $\bar{\Delta}_5 \xrightarrow{\mathsf{unify}\text{-}\beta} S_5$. By I.H., there exists P'_5 s.t.
$\quad \mathsf{readback}([S_5]\,P_5) = P'_5$,
$\quad \mathsf{term}(P'_5) = M_3[x := M_2]$, and
$\quad \mathsf{constraint}(P'_5) = [S_5]\,\Delta'_5$.
P'_5 derives the following judgement:
$(P'_5 \triangleright M_3[x := M_2]) : \langle [S_5]\,A_5 \vdash [S_5]\,\mathsf{a}_0 \rangle \;/\; [S_5]\,\Delta'_5$.
Let $M_6 = (\lambda x.M_4) \mathbin{@} M_2$ and $P_6 = \mathsf{initial}(M_4)$.
P_6 derives the following judgement:
$(M_6 \triangleright P_6) : \langle A_6 \vdash \mathsf{a}_0 \rangle \;/\; \Delta_6$, where
$\quad A_6 = \mathsf{e}_1\,\mathsf{e}_0\,A_4[x \mapsto \omega] \mathbin{\dot\cap} \mathsf{e}_2\,A_2$,
$\quad \Delta_6 = \bar{\Delta}_6 \mathbin{\dot\cap} \Delta'_6$
$\quad \Delta'_6 = \mathsf{e}_1\,\mathsf{e}_0\,\Delta_4 \mathbin{\dot\cap} \mathsf{e}_2\,\Delta_2$, and
$\quad \bar{\Delta}_6 = \mathsf{e}_1\,(\mathsf{e}_0\,\tau'_4 \to \mathsf{e}_0\,\tau_3) \leq \mathsf{e}_2\,\tau_2 \to \mathsf{a}_0$.
$$\text{Let } S_6 = \begin{array}{r|l} & (\mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_0\,E_4\,,\, \mathsf{e}_1/\mathsf{e}_0/S'_4) \\ ; & (\mathsf{a}_0 := [S'_4]\,\tau_4\,,\, \mathsf{e}_1 := \mathsf{e}_0 := \boxdot). \end{array}$$
Since $\tau'_4 \xrightarrow{\mathsf{extract}} E_4$ and $\tau'_4 \xrightarrow{:=\tau_2} S'_4$, $\bar{\Delta}_6 \xrightarrow{\mathsf{unify}\text{-}\beta} S_6$. By I.H., there exists P'_6 s.t.
$\quad \mathsf{readback}([S_6]\,P_6) = P'_6$,
$\quad \mathsf{term}(P'_6) = M_4[x := M_2]$, and
$\quad \mathsf{constraint}(P'_6) = [S_6]\,\Delta'_6$.
P'_6 derives the following judgement:
$(P'_6 \triangleright M_4[x := M_2]) : \langle [S_6]\,A_6 \vdash [S_6]\,\mathsf{a}_0 \rangle \;/\; [S_6]\,\Delta'_6$.
Let $P' = \mathsf{e}_1\,P'_4 \mathbin{@} \mathsf{e}_2\,P'_6$. P' derives this judgement:
$(M' \triangleright P') : \langle A' \vdash \tau' \rangle \;/\; \Delta''$ where
$M' = (M_3[x := M_2]) \mathbin{@} (M_4[x := M_2])$
$\quad = (M_3 \mathbin{@} M_4)[x := M_2]$
$\quad = M_1[x := M_2]$
$$\begin{array}{rr|l} A' = & & \mathsf{e}_1\,[S_5]\,A_5 \mathbin{\dot\cap} \mathsf{e}_2\,[S_6]\,A_6 \\ = & & \mathsf{e}_1\,(([S'_3]\,A_3)[x \mapsto \omega] \mathbin{\dot\cap} [E_3]\,A_2) \\ & \dot\cap & \mathsf{e}_2\,(([S'_4]\,A_4)[x \mapsto \omega] \mathbin{\dot\cap} [E_4]\,A_2) \\ = & & (\mathsf{e}_1\,[S'_3]\,A_3)[x \mapsto \omega] \mathbin{\dot\cap} \mathsf{e}_1\,[E_3]\,A_2 \\ & \dot\cap & (\mathsf{e}_2\,[S'_4]\,A_4)[x \mapsto \omega] \mathbin{\dot\cap} \mathsf{e}_2\,[E_4]\,A_2 \\ = & & (\mathsf{e}_1\,[S'_3]\,A_3 \mathbin{\dot\cap} \mathsf{e}_2\,[S'_4]\,A_4)[x \mapsto \omega] \\ & \dot\cap & \mathsf{e}_1\,[E_3]\,A_2 \mathbin{\dot\cap} \mathsf{e}_2\,[E_4]\,A_2 \\ = & & [S]\,A \end{array}$$
$\tau' = \mathsf{e}_1\,[S_5]\,\mathsf{a}_0 \mathbin{\dot\cap} \mathsf{e}_2\,[S_6]\,\mathsf{a}_0$
$\quad = \mathsf{e}_1\,[S'_3]\,\mathsf{a}_0 \mathbin{\dot\cap} \mathsf{e}_2\,[S'_4]\,\mathsf{a}_0$
$\quad = [S]\,\mathsf{a}_0$
$\Delta'' = \mathsf{e}_1\,[S_5]\,\Delta'_5 \mathbin{\dot\cap} \mathsf{e}_2\,[S_6]\,\Delta'_6$
$\quad = \mathsf{e}_1\,([S'_3]\,\Delta_3 \mathbin{\dot\cap} [E_3]\,\Delta_2) \mathbin{\dot\cap} \mathsf{e}_2\,([S'_4]\,\Delta_4 \mathbin{\dot\cap} [E_4]\,\Delta_2)$
$\quad = \mathsf{e}_1\,[S'_3]\,\Delta_3 \mathbin{\dot\cap} \mathsf{e}_1\,[E_3]\,\Delta_2 \mathbin{\dot\cap} \mathsf{e}_2\,[S'_4]\,\Delta_4 \mathbin{\dot\cap} \mathsf{e}_2\,[E_4]\,\Delta_2$
$\quad = \mathsf{e}_1\,[S'_3]\,\Delta_3 \mathbin{\dot\cap} \mathsf{e}_2\,[S'_4]\,\Delta_4 \mathbin{\dot\cap} \mathsf{e}_1\,[E_3]\,\Delta_2 \mathbin{\dot\cap} \mathsf{e}_2\,[E_4]\,\Delta_2$
$\quad = [S]\,\Delta'$ $\square$

B. IMPROVED READBACK

A slight modification to readback, which we have implemented, allows it to operate after unify-@ has been used. We present this variation because it may be of interest, and also because it is mentioned by an example in the supplementary material to the submission (appendix C.2). The first case of the definition of readback is modified so that

$$\begin{array}{l} \mathsf{readback}(A, \mathsf{a}_0, \omega) \\ \quad = x^{:\tau} \mathbin{@} Q_1 \mathbin{@} \cdots \mathbin{@} Q_n \\ \quad \text{if } A = \vec{e}_1\,\mathsf{e}_2\,A_1 \mathbin{\dot\cap} \cdots \mathbin{\dot\cap} \vec{e}_n\,\mathsf{e}_2\,A_n \mathbin{\dot\cap} (x : \tau) \\ \quad \text{and } \tau = \vec{e}_1\,\mathsf{e}_2\,\tau_1 \to \cdots \to \vec{e}_n\,\mathsf{e}_2\,\tau_n \to \mathsf{a}_0 \\ \quad \text{and } Q_1 = \vec{e}_1\,\mathsf{e}_2\,\mathsf{readback}(A_1, \tau_1, \omega) \\ \qquad \cdots \\ \qquad Q_n = \vec{e}_n\,\mathsf{e}_2\,\mathsf{readback}(A_n, \tau_n, \omega) \end{array}$$

where $\vec{e}_n = \epsilon$ and if $1 \leq i < n$ then $\vec{e}_i = \mathsf{e}_1 \cdot \vec{e}_{i+1}$.

This new definition is quite sensitive and only works for the rightmost/innermost processing of unify-@ constraints and only provided all unify-β constraints are solved first. We are working on a more robust version.

C. SUPPLEMENTARY MATERIAL

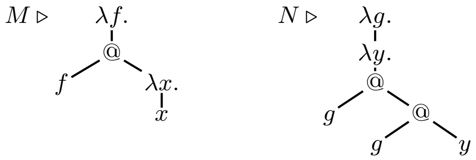
This supplementary material is collected here for the reader's convenience. It not needed as part of the paper, as it can be obtained elsewhere:

- Appendix C.1 is mostly taken from the paper introducing System E [6]. There are some minor improvements and the content has been specialized to the subset of System E used in this paper.
- Appendix C.2 contains LaTeX code that is automatically generated by the web interface to our implementation of type inference (`http://www.macs.hw.ac.uk/ultra/compositional-analysis/system-e/`).

C.1 A Brief Tutorial on Expansion and Expansion Variables

The essential notions of System E are expansion and expansion variables (E-variables) and System E's support for them is one of its main advantages over previous work. This section informally introduces the need for expansion and illustrates how E-variables help. The concepts of expansion and expansion variables are still unfamiliar to many people, but they are vital for this paper.

Consider two typing derivations structured as follows, where λ and @ represent appropriate typing rule uses:



In a standard intersection type system, the derived result types could be these:

$$M : \underbrace{((\alpha \to \alpha) \to \beta)}_{\tau_1} \to \beta$$

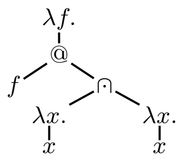
$$N : \underbrace{((\beta' \to \gamma') \sqcap (\alpha' \to \beta')) \to (\alpha' \to \gamma')}_{\tau_2}$$

In order to type the application $M \mathbin{@} N$, we must somehow "unify" τ_1 and τ_2. We could first unify the types $(\beta' \to \gamma')$ and $(\alpha' \to \beta')$ to collapse the intersection type $(\beta' \to \gamma') \sqcap (\alpha' \to \beta')$, but this would lose information and is not the approach we want to follow for full flexibility. Historically, in intersection type systems the solution has been to do *expansion* [9] on the result type of M:

$$M : (((\alpha_1 \to \alpha_1) \sqcap (\alpha_2 \to \alpha_2)) \to \beta) \to \beta \tag{1}$$

Then, the substitution $(\alpha_1 := \alpha', \beta' := \alpha', \alpha_2 := \alpha', \gamma' := \alpha', \beta := \alpha' \to \alpha')$ makes the application $M \mathbin{@} N$ typable.

What justified the expansion we did to the result type of M? The expansion operation above effectively altered the typing derivation for M by inserting a use of intersection introduction at a nested position in the previous derivation, transforming it into the following new derivation, where $\sqcap$ marks a use of the intersection-introduction typing rule:



Expansion variables are a new technology for simplifying expansion and making it easier to implement and reason about. Expansion variables are *placeholders* for unknown uses of other typing rules, such as intersection introduction, which are propagated into the types and the type constraints used by type inference algorithms. The E-variable-introduction typing rule works like this (in traditional notation, not as stated later in the paper), where "$e\,A$" applies the E-variable e to every type in the type environment A:

$$(\textsf{E-variable})\,\frac{A \vdash M : \tau}{e\,A \vdash M : e\,\tau}$$

Type-level substitutions in System E substitute types for type variables, and *expansions* for expansion variables. An expansion is a piece of syntax standing for some number of uses of typing rules that act uniformly on every type in a judgement and do not change the judgement's untyped term. The most trivial expansion, $\boxdot$, is the identity. Intersection types can also be introduced by expansion; for example, given $M : (e\,(\alpha \to \alpha) \to \beta) \to \beta$, applying the substitution $e := \boxdot \sqcap \boxdot$ to this typing yields $M : ((\alpha \to \alpha) \sqcap (\alpha \to \alpha) \to \beta) \to \beta$. Substitutions are also a form of expansion; if we apply the substitution $e := (\alpha := \alpha_1) \sqcap (\alpha := \alpha_2)$ to M, we get the expanded typing given for M in (1) above.

Including substitutions as a form of expansion makes expansion variables effectively establish namespaces which can be manipulated separately. For example, applying the substitution $(e_1 := (\alpha := \tau_1),\ e_2 := (\alpha := \tau_2))$ to the type $(e_1\,\alpha) \sqcap (e_2\,\alpha)$ yields the type $\tau_1 \sqcap \tau_2$.

The syntactic expansion ω (introduced as expansion syntax in [5]) is also included, along with a typing rule that assigns the type ω (introduced in [9]) to any term. If we apply the substitution $e := \omega$ to the typing of M above, inside the typing derivation the result type for $\lambda x.x$ becomes ω. Operationally, a term which has type ω can only be passed around and must eventually be discarded.

Types in System E are by default *linear*. For example, $(\alpha_1 \to \alpha_1) \Cap (\alpha_2 \to \alpha_2)$ is the type of a function to be used exactly twice, once at type $\alpha_1 \to \alpha_1$, and once at $\alpha_2 \to \alpha_2$.

Because it is sufficient to present the results of this paper, we consider only the !-free fragment of System E, in which all types are linear. The full System E also has a ! type constructor for creating non-linear types, and has ! as an additional case for expansion, thereby allowing uses of ! to be inserted by constraint solving.

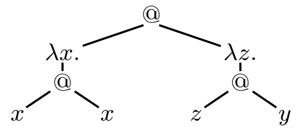
The use of expansions pervades every part of System E's design. The main syntactic sorts are types, typing constraints, skeletons (System E's proof terms), and expansions (and also subtyping proof terms, not shown in this paper). Each sort has cases for E-variable application, intersection, and ω (and ! in the full system). Other derived notions such as type environments also in effect support each of these operations. This common structure makes System E work.

C.2 Full Example of Type Inference

This appendix shows an example of type inference, presenting output produced by our implementation, which has the option of writing bits of LaTeX code suitable for inclusion in LaTeX documents. We analyze the term M:

$$M = (\lambda x.x \,@\, x) \,@\, (\lambda z.z \,@\, y)$$

The term M, drawn as a tree:



For compactness, uses of result type subtyping which add a singular constraint that is solved have been omitted from skeletons in the rest of this section. However, these solved constraints are included in judgements.
The following diagram sums up the relations between the entities presented in the rest of this section:

$$\begin{tikzcd}
& S_1 & & S_2 & & S_3 & & S_4 & \\
Q_0 \arrow[ur, "\text{unify-}\beta\text{-LO}"] \arrow[d, "\text{readback}"] & \langle S_1, Q_0\rangle \arrow[l, "\pi_2"'] \arrow[u, "\pi_1"'] \arrow[r, "{[\cdot]\cdot}"] & Q_1 \arrow[ur, "\text{unify-}\beta\text{-LO}"] \arrow[d, "\text{readback}"] & \langle S_2, Q_1\rangle \arrow[l, "\pi_2"'] \arrow[u, "\pi_1"'] \arrow[r, "{[\cdot]\cdot}"] & Q_2 \arrow[ur, "\text{unify-}\beta\text{-LO}"] \arrow[d, "\text{readback}"] & \langle S_3, Q_2\rangle \arrow[l, "\pi_2"'] \arrow[u, "\pi_1"'] \arrow[r, "{[\cdot]\cdot}"] & Q_3 \arrow[ur, "\text{unify-@-RI}"] \arrow[d, "\text{readback}"] & \langle S_4, Q_3\rangle \arrow[l, "\pi_2"'] \arrow[u, "\pi_1"'] \arrow[r, "{[\cdot]\cdot}"] & Q_4 \arrow[d, "\text{readback}'"] \\
P_0 \arrow[d, bend left, "\text{term}"] & & P_1 \arrow[d, bend left, "\text{term}"] & & P_2 \arrow[d, bend left, "\text{term}"] & & P_3 \arrow[d, bend left, "\text{term}"] & & P_4 \arrow[dll, "\text{term}"] \\
M \arrow[u, bend left, "\text{initial}"] \arrow[rr, "\beta\text{-LO}"] & & M_1 \arrow[u, bend left, "\text{initial}"] \arrow[rr, "\beta\text{-LO}"] & & M_2 \arrow[u, bend left, "\text{initial}"] \arrow[rr, "\beta\text{-LO}"] & & M_3 \arrow[u, bend left, "\text{initial}"] & &
\end{tikzcd}$$

where

- $\text{unify-}\beta\text{-LO} = \text{constraint}; \text{filter-}\beta; \text{LO}; \text{unify-}\beta$,
- $\text{unify-@-RI} = \text{constraint}; \text{RI}; \text{unify-@}$,
- β-LO is the least relation such that $C^{\text{lo}}[(\lambda x.M_1) \,@\, M_2] \xrightarrow{\beta\text{-LO}} C^{\text{lo}}[M_1[x := M_2]]$, and
- $\text{readback}'$ denotes the modified version of readback given in the next section.

C.2.1 Initial Skeleton

The inference process starts by picking an initial skeleton $Q_0 = P_0 = \text{initial}(M)$:

$$Q_0 = (\mathsf{e}_1\,(\lambda x.\,\mathsf{e}_0\,((\mathsf{e}_1\,(x^{:\mathsf{a}_0})^{:\mathsf{e}_2\,\mathsf{a}_0\to\mathsf{a}_0})\,@\,\mathsf{e}_2\,(x^{:\mathsf{a}_0})))^{:\mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0\to\mathsf{e}_0\,\mathsf{a}_0)\to\mathsf{a}_0})\,@\,\mathsf{e}_2\,(\lambda z.\,\mathsf{e}_0\,((\mathsf{e}_1\,(z^{:\mathsf{a}_0})^{:\mathsf{e}_2\,\mathsf{a}_0\to\mathsf{a}_0})\,@\,\mathsf{e}_2\,(y^{:\mathsf{a}_0})))$$

The judgement Q_0 derives is:

$$\begin{array}{l}
(M \rhd Q_0) : \langle A_0 \vdash \tau_0\rangle \,/\, \Delta_0, \text{where} \\
\quad A_0(y) = \mathsf{e}_2\,\mathsf{e}_0\,\mathsf{e}_2\,\mathsf{a}_0 \\
\quad \tau_0 = \mathsf{a}_0 \\
\quad \Delta_0 = \quad \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \Cap \mathsf{e}_0\,\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \leq \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0 \\
\qquad\qquad \Cap\, \mathsf{e}_1\,\mathsf{e}_0\,(\mathsf{e}_1\,\mathsf{a}_0 \leq \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \\
\qquad\qquad \Cap\, \mathsf{e}_2\,\mathsf{e}_0\,(\mathsf{e}_1\,\mathsf{a}_0 \leq \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0)
\end{array}$$

The readback P_0 of the judgement derived by Q_0 is:

$$\begin{array}{l}
\text{readback}(A_0, \tau_0, \Delta_0) = P_0 = \\
\quad (\mathsf{e}_1\,(\lambda x.\,\mathsf{e}_0\,((\mathsf{e}_1\,(x^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0)\,@\,\mathsf{e}_2\,(x^{:\mathsf{a}_0}))) : \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0)\,@\,\mathsf{e}_2\,(\lambda x.\,\mathsf{e}_0\,((\mathsf{e}_1\,(x^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0)\,@\,\mathsf{e}_2\,(y^{:\mathsf{a}_0})))
\end{array}$$

Note that, since we have quotiented skeletons and terms by α-equivalence (renaming of bound variables), $P_0 = Q_0$ holds.
The untyped term associated with P_0 is:

$$\text{term}(P_0) = (\lambda x.x \,@\, x) \,@\, (\lambda x.x \,@\, y)$$

Of course, $\text{term}(P_0) = M$.

C.2.2 *Step 1 (use of* unify-β*)*

The leftmost-outermost unsolved constraint of Q_0 to which unify-β applies is:

$$\dot{\Delta}_1 = \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \mathbin{\dot\cap} \mathsf{e}_0\,\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \mathrel{\dot\leq} \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0$$

The result of unify-β applied to $\dot{\Delta}_1$ is:

$$\dot{\Delta}_1 \xrightarrow{\text{unify-}\beta} S_1 = \begin{array}{r|l} & \begin{array}{r|l} & \mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_0\,(\mathsf{e}_1\,\boxdot \mathbin{\dot\cap} \mathsf{e}_2\,\boxdot) \\ , & \begin{array}{r|l} & \mathsf{e}_1 := \mathsf{e}_1\,(\begin{array}{r|l} & \mathsf{e}_0 := \mathsf{e}_0\,(\mathsf{e}_1 := \mathsf{e}_1\,(\mathsf{a}_0 := \mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0\,,\boxdot)\,,\boxdot\,;\,\mathsf{e}_2 := \mathsf{e}_2\,(\mathsf{a}_0 := \mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0\,,\boxdot)\,,\boxdot) \\ , & \boxdot) \end{array} \\ , & \boxdot \end{array} \end{array} \\ ; & \mathsf{a}_0 := \mathsf{a}_0\,,\,\mathsf{e}_1 := \mathsf{e}_0 := \boxdot\,,\boxdot\,,\boxdot \end{array}$$

The skeleton $Q_1 = [S_1]\,Q_0$ is:

$$Q_1 = \begin{array}{r|l} & (\lambda x.\,(\mathsf{e}_1\,(x^{:\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0}) : \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(x^{:\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0})) \\ @ & \mathsf{e}_1\,(\lambda z.\,\mathsf{e}_0\,((\mathsf{e}_1\,(z^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}))) \mathbin{\dot\cap} \mathsf{e}_2\,(\lambda z.\,\mathsf{e}_0\,((\mathsf{e}_1\,(z^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}))) \end{array}$$

The judgement Q_1 derives is:

$$\begin{aligned} &(M \rhd Q_1) : \langle A_1 \vdash \tau_1 \rangle \;/\; \Delta_1, \text{where} \\ &\quad A_1(y) = \mathsf{e}_1\,\mathsf{e}_0\,\mathsf{e}_2\,\mathsf{a}_0 \mathbin{\dot\cap} \mathsf{e}_2\,\mathsf{e}_0\,\mathsf{e}_2\,\mathsf{a}_0 \\ &\quad \tau_1 = \mathsf{a}_0 \\ &\quad \Delta_1 = \quad \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \mathrel{\dot\leq} \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0 \\ &\qquad\quad \mathbin{\dot\cap} \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \mathbin{\dot\cap} \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0 \mathrel{\dot\leq} \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \mathbin{\dot\cap} \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0 \\ &\qquad\quad \mathbin{\dot\cap} \mathsf{e}_1\,\mathsf{e}_0\,(\mathsf{e}_1\,\mathsf{a}_0 \mathrel{\dot\leq} \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \\ &\qquad\quad \mathbin{\dot\cap} \mathsf{e}_2\,\mathsf{e}_0\,(\mathsf{e}_1\,\mathsf{a}_0 \mathrel{\dot\leq} \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \end{aligned}$$

The readback P_1 of the judgement derived by Q_1 is:

$$\begin{aligned} &\mathsf{readback}(A_1, \tau_1, \Delta_1) = P_1 = \\ &\quad (\mathsf{e}_1\,(\lambda x.\,\mathsf{e}_0\,((\mathsf{e}_1\,(x^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}))) : \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(\lambda x.\,\mathsf{e}_0\,((\mathsf{e}_1\,(x^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}))) \end{aligned}$$

The untyped term associated with P_1 is:

$$\mathsf{term}(P_1) = (\lambda x.x \mathbin{@} y) \mathbin{@} (\lambda x.x \mathbin{@} y)$$

Note that $\mathsf{term}(P_0) \xrightarrow{\beta\text{-LO}} \mathsf{term}(P_1)$.

C.2.3 *Step 2 (use of* unify-β*)*

The leftmost-outermost unsolved constraint of Q_1 to which unify-β applies is:

$$\dot{\Delta}_2 = \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \mathrel{\dot\leq} \mathsf{e}_2\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0$$

The result of unify-β applied to $\dot{\Delta}_2$ is:

$$\dot{\Delta}_2 \xrightarrow{\text{unify-}\beta} S_2 = \begin{array}{r|l} & \mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_0\,\mathsf{e}_1\,\boxdot\,,\,\mathsf{e}_1 := \mathsf{e}_1\,(\mathsf{e}_0 := \mathsf{e}_0\,(\mathsf{e}_1 := \mathsf{e}_1\,(\mathsf{a}_0 := \mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0\,,\boxdot)\,,\boxdot)\,,\boxdot)\,,\boxdot \\ ; & \mathsf{a}_0 := \mathsf{a}_0\,,\,\mathsf{e}_1 := \mathsf{e}_0 := \boxdot\,,\boxdot\,,\boxdot \end{array}$$

The skeleton $Q_2 = [S_2]\,Q_1$ is:

$$Q_2 = \begin{array}{r|l} & (\lambda x.\,(x^{:\mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0}) \mathbin{@} \mathsf{e}_1\,(x^{:\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0})) \\ @ & (\lambda z.\,(\mathsf{e}_1\,(z^{:\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0})) \mathbin{\dot\cap} \mathsf{e}_1\,(\lambda z.\,\mathsf{e}_0\,((\mathsf{e}_1\,(z^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}))) \end{array}$$

The judgement Q_2 derives is:

$$\begin{aligned} &(M \rhd Q_2) : \langle A_2 \vdash \tau_2 \rangle \;/\; \Delta_2, \text{where} \\ &\quad A_2(y) = \mathsf{e}_2\,\mathsf{a}_0 \mathbin{\dot\cap} \mathsf{e}_1\,\mathsf{e}_0\,\mathsf{e}_2\,\mathsf{a}_0 \\ &\quad \tau_2 = \mathsf{a}_0 \\ &\quad \Delta_2 = \quad \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0 \mathrel{\dot\leq} \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0 \\ &\qquad\quad \mathbin{\dot\cap} (\mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0) \mathbin{\dot\cap} \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0 \mathrel{\dot\leq} (\mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0) \mathbin{\dot\cap} \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \to \mathsf{a}_0 \\ &\qquad\quad \mathbin{\dot\cap} \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \mathrel{\dot\leq} \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0 \\ &\qquad\quad \mathbin{\dot\cap} \mathsf{e}_1\,\mathsf{e}_0\,(\mathsf{e}_1\,\mathsf{a}_0 \mathrel{\dot\leq} \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \end{aligned}$$

The readback P_2 of the judgement derived by Q_2 is:

$$\begin{aligned} &\mathsf{readback}(A_2, \tau_2, \Delta_2) = P_2 = \\ &\quad (\mathsf{e}_1\,(\lambda x.\,\mathsf{e}_0\,((\mathsf{e}_1\,(x^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}))) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}) \end{aligned}$$

The untyped term associated with P_2 is:

$$\mathsf{term}(P_2) = (\lambda x. x \mathbin{@} y) \mathbin{@} y$$

Note that $\mathsf{term}(P_1) \xrightarrow{\beta\text{-LO}} \mathsf{term}(P_2)$.

C.2.4 Step 3 (use of unify-β)

The leftmost-outermost unsolved constraint of Q_1 to which unify-β applies is:

$$\dot{\Delta}_3 = \mathsf{e}_1\,(\mathsf{e}_0\,\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{e}_0\,\mathsf{a}_0) \preceq \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0$$

The result of unify-β applied to $\dot{\Delta}_3$ is:

$$\dot{\Delta}_3 \xrightarrow{\text{unify-}\beta} S_3 = \mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_0\,\mathsf{e}_1\,\boxdot\,,\, \mathsf{e}_1 := \mathsf{e}_1\,(\mathsf{e}_0 := \mathsf{e}_0\,(\mathsf{e}_1 := \mathsf{e}_1\,(\mathsf{a}_0 := \mathsf{a}_0\,,\boxdot)\,,\boxdot)\,,\boxdot)\,,\boxdot\,;\, \mathsf{a}_0 := \mathsf{a}_0\,,\, \mathsf{e}_1 := \mathsf{e}_0 := \boxdot\,,\boxdot\,,\boxdot$$

The skeleton $Q_3 = [S_3]\,Q_2$ is:

$$Q_3 = (\lambda x.\,(x^{:(\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0)\to \mathsf{a}_0}) \mathbin{@} (x^{:\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0})) \mathbin{@} (\lambda z.\,(z^{:\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0}) \mathbin{@} \mathsf{e}_1\,(y^{:\mathsf{a}_0})) \mathbin{\dot\cap} (\lambda z.\,(\mathsf{e}_1\,(z^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}))$$

The judgement Q_3 derives is:

$$\begin{array}{l}
(M \triangleright Q_3) : \langle A_3 \vdash \tau_3\rangle \;/\; \Delta_3, \text{where}\\
\quad A_3(y) = \mathsf{e}_1\,\mathsf{a}_0 \mathbin{\dot\cap} \mathsf{e}_2\,\mathsf{a}_0\\
\quad \tau_3 = \mathsf{a}_0\\
\quad \Delta_3 = \quad (\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0 \preceq (\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0\\
\qquad\qquad \mathbin{\dot\cap} ((\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0) \mathbin{\dot\cap} (\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0 \preceq ((\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0) \mathbin{\dot\cap} (\mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0\\
\qquad\qquad \mathbin{\dot\cap} \mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0 \preceq \mathsf{e}_1\,\mathsf{a}_0 \to \mathsf{a}_0\\
\qquad\qquad \mathbin{\dot\cap} \mathsf{e}_1\,\mathsf{a}_0 \preceq \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0
\end{array}$$

The readback P_3 of the judgement derived by Q_3 is:

$$\mathsf{readback}(A_3, \tau_3, \Delta_3) = P_3 = (\mathsf{e}_1\,(y^{:\mathsf{a}_0}) : \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0})$$

The untyped term associated with P_3 is:

$$\mathsf{term}(P_3) = y \mathbin{@} y$$

Note that $\mathsf{term}(P_2) \xrightarrow{\beta\text{-LO}} \mathsf{term}(P_3)$, and $\mathsf{term}(P_3)$ is a β-normal form.

C.2.5 Step 4 (use of unify-@)

At this step, Q_3 has no singular constraint to which unify-@ applies, so we switch to using unify-@ in order to solve the remaining constraints. The rightmost-innermost unsolved constraint of Q_3 to which unify-@ applies is:

$$\dot{\Delta}_4 = \mathsf{e}_1\,\mathsf{a}_0 \preceq \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0$$

The result of unify-@ applied to $\dot{\Delta}_4$ is:

$$\dot{\Delta}_4 \xrightarrow{\text{unify-}@} S_4 = \mathsf{e}_1 := \mathsf{a}_0 := \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0,\; \mathsf{e}_1 := \mathsf{e}_1\,\mathsf{e}_1\,\boxdot,\; \mathsf{e}_2 := \mathsf{e}_1\,\mathsf{e}_2\,\boxdot,\; \boxdot,\; \boxdot$$

The skeleton $Q_4 = [S_4]\,Q_3$ is:

$$Q_4 = (\lambda x.\,(x^{:((\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0)\to \mathsf{a}_0)\to \mathsf{a}_0}) \mathbin{@} (x^{:(\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0)\to \mathsf{a}_0})) \mathbin{@} (\lambda z.\,(z^{:(\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0)\to \mathsf{a}_0}) \mathbin{@} (y^{:\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0})) \mathbin{\dot\cap} (\lambda z.\,(z^{:\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0}) \mathbin{@} \mathsf{e}_2\,(y^{:\mathsf{a}_0}))$$

The judgement Q_4 derives is:

$$\begin{array}{l}
(M \triangleright Q_4) : \langle A_4 \vdash \tau_4\rangle \;/\; \Delta_4, \text{where}\\
\quad A_4(y) = (\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \mathbin{\dot\cap} \mathsf{e}_2\,\mathsf{a}_0\\
\quad \tau_4 = \mathsf{a}_0\\
\quad \Delta_4 = \quad ((\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0) \to \mathsf{a}_0 \preceq ((\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0) \to \mathsf{a}_0\\
\qquad\qquad \mathbin{\dot\cap} (((\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0) \to \mathsf{a}_0) \mathbin{\dot\cap} ((\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0) \to \mathsf{a}_0 \preceq (((\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0) \to \mathsf{a}_0) \mathbin{\dot\cap} ((\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0) \to \mathsf{a}_0\\
\qquad\qquad \mathbin{\dot\cap} (\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0 \preceq (\mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0) \to \mathsf{a}_0\\
\qquad\qquad \mathbin{\dot\cap} \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0 \preceq \mathsf{e}_2\,\mathsf{a}_0 \to \mathsf{a}_0
\end{array}$$

Note that Q_4 is solved.
With the extended definition of readback (implemented, included in the appendix of the paper), the readback P_4 of the judgement derived by Q_4 is the same as P_3:

$$\mathsf{readback}(A_4, \tau_4, \Delta_4) = P_4 = (y^{:\mathsf{a}_0 \to \mathsf{a}_0}) \mathbin{@} (y^{:\mathsf{a}_0})$$