Branching Types*

J. B. Wells†
Heriot-Watt University
Edinburgh

Christian Haack
Radboud Universiteit
Nijmegen

2006-10-30

Abstract

Although systems with intersection types have many unique capabilities, there has never been a fully satisfactory explicitly typed system with intersection types. We introduce and prove the basic properties of λ^{B}, a typed λ-calculus with *branching types* and types with quantification over *type selection parameters*. The new system λ^{B} is an explicitly typed system with the same expressiveness as a system with intersection types. Typing derivations in λ^{B} use branching types to squash together what would be separate parallel derivations in earlier systems with intersection types.

Part I
Informal Presentation

This part presents our new system λ^{B} and its motivation and implications in a way that we hope is understandable yet without requiring too many technical details. The technical presentation and formal statements are deferred to part II.

1 Background and Motivation

1.1 Intersection Types

Intersection types were independently invented near the end of the 1970s by Coppo and Dezani [CDC80] and Pottinger [Pot80]. Intersection types provide type polymorphism by listing type instances, differing from the more widely used $\forall$-quantified types [Gir72, Rey74], which provide type polymorphism by giving a type scheme that can be instantiated into various type instances via different substitutions of types for quantified type variables. The original motivation was for analyzing and/or synthesizing λ-models as well as in analyzing normalization properties, but over the last twenty years the scope of research on intersection types has broadened. Van Bakel has written a good summary of the early research [vB95].

Intersection types have many unique advantages over $\forall$-quantified types. First, they can characterize the behavior of λ-terms more precisely, and can be used to express exactly the results of many program analyses [PP01, AT00, WDMT97, WDMT02]. In particular, intersection types can give more detailed information about all the contexts in which a

*This work was partly supported by EPSRC grants GR/L 36963 and GR/R 41545/01, NATO grant CRG 971607, NSF grants CCR 9113196, 9417382, 9988529, and EIA 9806745, and Sun Microsystems equipment grant EDUD-7826-990410-US.

†Corresponding author. E-mail: `jbw@cee.hw.ac.uk`. Web: `http://www.cee.hw.ac.uk/~jbw/`.

function is used. Type polymorphism with intersection types is also more flexible. For example, Urzyczyn [Urz97] proved the λ-term

$$(\lambda x.z(x(\lambda fu.fu))(x(\lambda vg.gv)))(\lambda y.yyy)$$

to be untypable in the system F_ω, considered to be the most powerful type system with $\forall$-quantifiers measured by the set of pure λ-terms it can type. In contrast, this λ-term is typable with intersection types satisfying the *rank-3* restriction [KW99]. Second, better results for automated type inference (ATI) have also been obtained for intersection types. ATI for type systems with $\forall$-quantifiers that are more powerful than the very-restricted Hindley/Milner system is a murky area, and it has been proven for many such type systems that ATI algorithms can not be both complete and terminating [KW94, Wel96, Wel99, Urz97]. In contrast, ATI algorithms have been proven complete and terminating for the rank-k restriction for every finite k for several systems with intersection types [KW99, KMTW99]. Finally, intersection type systems often have the *principal typing* property, which facilitates modular program analysis [Wel02, KW99, Ban97, Jim95].

To obtain the advantages mentioned above, we use intersection types in typed intermediate languages (TILs) used in compilers. Using a TIL increases reliability of compilation and can support useful type-directed program transformations. We use intersection types to support more accurate analyses (as mentioned above) such as polyvariant flow analyses. We also use them to carry out interesting type/flow-directed transformations in which functions are customized in multiple ways for different uses [DMTW97, TDMW97, DWM$^+$01b, DWM$^+$01a] that would be very difficult using $\forall$-quantified types. When using a TIL, it is important to regularly check that the intermediate program representation is in fact well typed. Provided this is done, the correctness of any analyses encoded in the types is maintained across transformations. For this purpose, it is important for a TIL to be *explicitly* typed, i.e., to have type information attached to internal nodes of the program representation. This is necessary both for efficiency and because program transformations can yield results outside the domain of ATI algorithms. Unfortunately, intersection types raise troublesome issues for having an explicitly typed representation. This is the main motivation for this paper.

1.2 The Trouble with the Intersection-Introduction Rule

The important feature of a system with intersection types is this rule:

$$\frac{E \vdash M : \sigma; \quad E \vdash M : \tau}{E \vdash M : \sigma \wedge \tau} \ (\wedge\text{-intro})$$

The proof terms are the same for both premises and the conclusion! No syntax is introduced. A system with this rule does not fit into the proofs-as-terms (PAT, a.k.a. propositions-as-types and Curry/Howard) correspondence, because it has proof terms that do not encode deductions. Unfortunately, this is inadequate for many needs, and there is an immediate dilemma in how to make a type-annotated variant of the system. The usual strategy fails immediately, e.g.:

$$\frac{E \vdash (\lambda x{:}\sigma.\, x) : (\sigma \to \sigma); \quad E \vdash (\lambda x{:}\tau.\, x) : (\tau \to \tau)}{E \vdash (\lambda x{:}\boxed{???}.\, x) : (\sigma \to \sigma) \wedge (\tau \to \tau)}$$

Where $\boxed{???}$ appears, what should be written?

This trouble is related to the fact that the $\wedge$ type constructor is not a truth-functional propositional connective, but rather one that depends on the proofs of the propositions it connects. Hindley showed that $\wedge$ does not correspond to traditional conjunction [Hin84], e.g., the type $\sigma \to \tau \to (\sigma \wedge \tau)$ is not inhabited. However, $\wedge$ has been shown to correspond to conjunction in a Relevant Logic [Ven94, DCGV97]. In the context of realizability, the

logical meaning of $\wedge$ has been called *strong conjunction* [Pot80, LE85, Min89, AB91, BM94]. A realizer of the ordinary conjunction $\sigma \mathbin{\&} \tau$ is a pair of a realizer of σ and a realizer of τ, but a realizer of the strong conjunction $\sigma \wedge \tau$ is a realizer of σ which is simultaneously a realizer of τ. (We use the non-standard symbol "&" for ordinary conjunction here merely to distinguish it from our use of "$\wedge$" for strong conjunction.) This requires a notion of equality on realizers, which goes beyond strict Brouwerism. Other logical connectives which have a proof-functional nature include *relevant implication* and *strong equivalence* [BM94].

1.3 Earlier Approaches

In the language Forsythe [Rey96], Reynolds annotates the binding of $(\lambda x.M)$ with a list of types, e.g., $(\lambda x{:}\sigma_1|\cdots|\sigma_n.\, M)$. If the abstraction body M is typable with a fixed type τ for each type σ_i for x, then the abstraction gets the type $(\sigma_1 \to \tau) \wedge \cdots \wedge (\sigma_n \to \tau)$. However, this approach can not handle dependencies between types of nested variable bindings, e.g., this approach can not give $K = (\lambda x.\lambda y.x)$ the type $\tau_K = (\sigma \to (\sigma \to \sigma)) \wedge (\tau \to (\tau \to \tau))$.

Pierce [Pie91] improves on Reynolds's approach by using a **for** construct which gives a type variable a finite set of types to range over, e.g., K can be annotated as $(\mathbf{for}\ \alpha \in \{\sigma,\tau\}.\lambda x{:}\alpha.\, \lambda y{:}\alpha.\, x)$ with the type τ_K. However, this approach can not represent some typings, e.g., it can not give the term $M_f = \lambda x.\lambda y.\lambda z.(xy, xz)$ the type $(((\alpha \to \delta) \wedge (\beta \to \epsilon)) \to \alpha \to \beta \to (\delta \times \epsilon)) \wedge ((\gamma \to \gamma) \to \gamma \to \gamma \to (\gamma \times \gamma))$. Pierce's approach could be extended to handle more complex dependencies if simultaneous type variable bindings were added, e.g., M_f could be annotated as:

$$\begin{array}{l}\mathbf{for}\ \{[\theta \mapsto \alpha, \kappa \mapsto \beta, \eta \mapsto \delta, \nu \mapsto \epsilon], [\theta \mapsto \gamma, \kappa \mapsto \gamma, \eta \mapsto \gamma, \nu \mapsto \gamma]\}. \\ \quad \lambda x : (\theta \to \eta) \wedge (\kappa \to \nu)\, .\, \lambda y : \theta\, .\, \lambda z : \kappa\, .\, (xy, xz)\end{array}$$

Even this extension of Pierce's approach would still not meet our needs. First, this approach needs intersection types to be associative, commutative, and idempotent (ACI). Recent research suggests that non-ACI intersection types are needed to faithfully encode flow analyses [AT00]. Second, this approach arranges the type information inconveniently because it must be found from enclosing type variable bindings by a tree-walking process. This is bad for flow-based transformations, which reference arbitrary subterms from distant locations. Third, reasoning about typed terms in this approach is not compositional. It is not possible to look at an arbitrary subterm independently and determine its type solely from the type annotations within that subterm.

The approach of λ^{CIL} [WDMT97, WDMT02] is essentially to write the typing derivations as terms, e.g., K can be "annotated" as $\bigwedge((\lambda x{:}\sigma.\, \lambda y{:}\sigma.\, x), (\lambda x{:}\tau.\, \lambda y{:}\tau.\, x))$ in order to have the type τ_K. Here $\bigwedge(M, N)$ is a *virtual* tuple where the type erasure of M and N must be the same. In λ^{CIL}, subterms of an untyped term can have many disjoint representatives in a corresponding typed term. This makes it tedious and time-consuming to implement common program transformations, because *parallel contexts* must be used whenever subterms are transformed.

Venneri succeeded in completely removing the ($\wedge$-intro) rule from a type system with intersection types, but this was for combinatory logic rather than the λ-calculus [Ven94, DCGV97], and the approach seems unlikely to be transferable to the λ-calculus. In Section 4, we will compare our approach to recent related work of Ronchi Della Rocca and Roversi [RDRR01], Capitani, Loreti, and Venneri [CLV01], and Liquori and Ronchi Della Rocca [LRDR05].

2 An Introductory Example

This section uses an example to introduce our approach to solving the problem of the intersection introduction rule with our new system λ^{B} of *branching types*. The example

is presented first in a traditional system of intersection types, then in an explicitly typed system of intersection types in the style of λ^{CIL}, and then in λ^{B}.

2.1 Record-like Syntax and Pseudo-Grammars

We give here some notation that will be used throughout this article and which is used in the examples in this part.

Let $\mathcal{I}$ be a fixed countably infinite set of *labels*. Let the letter I range over finite subsets of $\mathcal{I}$, and let the letters i, j, k, l, m, n, o, and p range over elements of $\mathcal{I}$. For non-empty and finite I, let $\{i = v_i\}^I$ denote the function $\{\,(i, v_i)\,|\,i \in I\,\}$.

The notation $\{i = v_i\}^I$ acts like a *record* whose field names are I. We use this record-like notation in what we call *pseudo-grammars*. Pseudo-grammars are to be interpreted as inductive definitions in the obvious way, but are not proper grammars because rather than defining proper syntax they define sets that contain embedded mathematical objects like finite functions.

This record-like notation is non-commutative in the sense that in general $\{i = \sigma,\ j = \tau\} \neq \{i = \tau,\ j = \sigma\}$. This record-like notation is non-associative in the sense that there do not exist any $\vec{i}$ and $\vec{j}$ such that $\{i_0 = \sigma,\ i_1 = \{i_2 = \tau,\ i_3 = \rho\}\} = \{j_0 = \sigma,\ j_1 = \tau,\ j_2 = \rho\}$.

Remark 2.1. We will use this record-like notation for types in a way that is not conventional, but highly motivated by practical experience including many years of work on the Church Project's experimental CIL compiler [WDMT97, WDMT02]. For example, an intersection type traditionally written as $\sigma \wedge \tau$ will instead be written as $\wedge\{i = \sigma,\ j = \tau\}$. This style gives us many of the benefits of making the intersection type constructor associative and commutative while avoiding the costs of actually doing so. One cost we avoid is needing to work modulo the rules for associativity and commutativity, which is significant because our system λ^{B} needs a different equational theory on types and combining them would be overcomplicated. Another cost we avoid is that usually with an associative and commutative operator a procedure for deciding equality must try all combinations. (Although we will use another equational theory (as mentioned just above), our equational theory has normal forms and thus lacks this difficulty.)

A similar alternative we rejected would be to use multisets of types; we did not do this because labeling the individual types is vital in defining λ^{B}'s equational theory on types. Also, using the same record-like syntax for presenting the more traditional intersection type systems makes the comparison with λ^{B} clearer.

We could have used sequences of types, which is equivalent to insisting each label set I is of the form $\{0, 1, 2, \ldots, n\}$. We did not do so because the freedom to choose arbitrary labels allows particular typing derivations to avoid unneeded coincidences. In other words, we can implement things so that the same label i appears in two places only if it *must* be the same in both places. This is invaluable for debugging and makes examples more readable. $\Box$

2.2 A Traditional System of Intersection Types and an Example

The sets $\mathsf{UntypedTerm}$ of *untyped terms* (of the pure λ-calculus) and IntTy of *intersection types* are defined by the following pseudo-grammars:

$$
\begin{array}{rcll}
x & \in & \mathsf{Var} & \\
\hat{M}, \hat{N} & \in & \mathsf{UntypedTerm} & ::= \ x \ \mid \ \lambda x.\hat{M} \ \mid \ \hat{M}\ \hat{N} \\
\alpha & \in & \mathsf{TyVar} & \\
\hat{\sigma}, \hat{\tau} & \in & \mathsf{IntTy} & ::= \ \alpha \ \mid \ \hat{\sigma} \to \hat{\tau} \ \mid \ \wedge\{i = \hat{\tau}_i\}^I
\end{array}
$$

The sets Var and TyVar are countably infinite sets of respectively term variables and type variables. *Environments* are finite functions from Var to IntTy. Let the letter $\hat{E}$ range over the set Env of environments. The typing rules are shown in Figure 1. This system is a fairly

standard intersection type system where the intersection type constructor is n-ary, non-associative, non-commutative, and non-idempotent. In this article, we refer to this system as λ^{I}.

$$(\mathsf{ax}) \qquad \frac{}{\hat{E} \vdash^{\mathsf{i}} x : \hat{E}(x)}$$

$$(\rightarrow_{\mathsf{i}}) \qquad \frac{\hat{E}[x \mapsto \hat{\sigma}] \vdash^{\mathsf{i}} \hat{M} : \hat{\tau}}{\hat{E} \vdash^{\mathsf{i}} \lambda x.\hat{M} : \hat{\sigma} \rightarrow \hat{\tau}}$$

$$(\rightarrow_{\mathsf{e}}) \qquad \frac{\hat{E} \vdash^{\mathsf{i}} \hat{M} : \hat{\sigma} \rightarrow \hat{\tau}; \quad \hat{E} \vdash^{\mathsf{i}} \hat{N} : \hat{\sigma}}{\hat{E} \vdash^{\mathsf{i}} \hat{M}\ \hat{N} : \hat{\tau}}$$

$$(\wedge_{\mathsf{i}}) \qquad \frac{\hat{E} \vdash^{\mathsf{i}} \hat{M} : \hat{\tau}_i \text{ for all } i \in I}{\hat{E} \vdash^{\mathsf{i}} \hat{M} : \wedge\{i = \hat{\tau}_i\}^I}$$

$$(\wedge_{\mathsf{e}}) \qquad \frac{\hat{E} \vdash^{\mathsf{i}} \hat{M} : \wedge\{i = \hat{\tau}_i\}^I}{\hat{E} \vdash^{\mathsf{i}} \hat{M} : \hat{\tau}_j} \quad \text{if } j \in I$$

Figure 1: Typing rules for the intersection type system λ^{I}

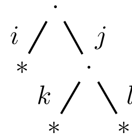
Example 2.2. Consider the following untyped term:

$$\hat{M} = \lambda x.\lambda y.\lambda z.\ z\ (\lambda w.\ w\ x\ y)$$

Among other possible typings, $\hat{M}$ can be given in λ^{I} the type $\wedge\{i = \hat{\tau},\ j = \hat{\sigma}\}$, where $\hat{\tau}$ and $\hat{\sigma}$ are:

$$\begin{array}{rcl}
\hat{\tau} & = & (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \hat{\tau}^z \rightarrow \beta \\
\hat{\tau}^z & = & (\hat{\tau}^w \rightarrow \beta) \rightarrow \beta \\
\hat{\tau}^w & = & (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta \\
\hat{\sigma} & = & \gamma \rightarrow \wedge\{m = \gamma \rightarrow \gamma, n = \beta \rightarrow \beta\} \rightarrow \hat{\sigma}^z \rightarrow \beta \\
\hat{\sigma}^z & = & \wedge\{k = \hat{\sigma}^w_k \rightarrow \beta, l = \hat{\sigma}^w_l \rightarrow \beta\} \rightarrow \beta \\
\hat{\sigma}^w_k & = & \gamma \rightarrow (\gamma \rightarrow \gamma) \rightarrow \beta \\
\hat{\sigma}^w_l & = & \gamma \rightarrow (\beta \rightarrow \beta) \rightarrow \beta
\end{array}$$

The reader might want to write out the derivation (tree of judgments) that $\hat{M}$ has type $\wedge\{i = \hat{\tau},\ j = \hat{\sigma}\}$. (The reader can also look ahead to Figure 3, which gives a term whose structure is isomorphic to that of this derivation.) The reader will notice that the derivation has two different subderivations for $\hat{M}$ — one proving that $\hat{M}$ has type $\hat{\tau}$ ("the i-subderivation") and the other one that it has type $\hat{\sigma}$ ("the j-subderivation"). Moreover, within the j-subderivation, there are two different subderivations for the subterm $(\lambda w.\, w\ x\ y)$ — one proving that it has type $(\hat{\sigma}^w_k \rightarrow \beta)$ ("the jk-subderivation") and the other one that it has type $(\hat{\sigma}^w_l \rightarrow \beta)$ ("the jl-subderivation"). Thus, there are altogether three different subderivations for the subterm $(\lambda w.\, w\ x\ y)$, one subderivation of the i-derivation and two subderivations of the j-derivation. We depict this sort of "branching structure" of the type derivation by the following tree:



The internal nodes of this tree correspond to uses of the intersection introduction rule $(\wedge_{\mathsf{i}})$. Throughout the paper we use $*$ for the leaves of these kinds of trees. In this case, $*$ corresponds to subderivations that do not contain any uses of the intersection introduction rule. We call this tree the *branching shape* of the type derivation. □

2.3 The Example in an Explicitly Typed System with Duplicates

Wells et al. [WDMT02] introduce the intersection type system λ^{CIL} (for "Church Intermediate Language") — a system with explicit type annotations. Here we present a small sublanguage of λ^{CIL} following the presentation of [DWM+01c], which uses a record-like syntax for intersection and union types. The set $\mathsf{DupTerm}$ of *explicitly typed terms with duplicates* is defined by the following pseudo-grammar:

$$\check{M}, \check{N} \in \mathsf{DupTerm} \quad ::= \quad x^{\hat{\tau}} \;\mid\; \lambda x^{\hat{\tau}}.\check{M} \;\mid \check{M}\ \check{N} \mid\; \wedge\{i = \check{M}_i\}^I \;\mid\; \pi_i^{\wedge}\ \check{M}$$

We define a partial function $|\cdot|$ from $\mathsf{DupTerm}$ to $\mathsf{UntypedTerm}$ that erases type annotations. The partial function is defined inductively by the following set of equations:

$$\begin{array}{rcl} |x^{\hat{\tau}}| &=& x \qquad\qquad |\lambda x^{\hat{\tau}}.\check{M}| \;=\; \lambda x.|\check{M}| \\ |\check{M}\ \check{N}| &=& |\check{M}|\ |\check{N}| \qquad |\pi_i^{\wedge}\ \check{M}| \;=\; |\check{M}| \\ |\wedge\{i = \check{M}_i\}^I| &=& |\check{M}_j|, \quad \text{if } j \in I \text{ and } |\check{M}_j| = |\check{M}_i| \neq \bot \text{ for all } i \in I \end{array}$$

Note that the function $|\cdot|$ is not total, by the last clause of the inductive definition. The typing rules are shown in Figure 2.

In λ^{CIL}, terms of the shape $\wedge\{i = \check{M}_i\}^I$ are *virtual records* that are syntax representing a use of the intersection introduction typing rule. Syntactically, there is no essential difference between virtual records and "proper" records. The difference is in their type introduction rules. The typing rule $(\wedge_{\mathsf{i}})$ for virtual records requires that the components of well typed virtual records must have equal type erasures. In other words, the components of a well typed virtual record all represent the same untyped term and differ only in their type annotations. Virtual records can be implemented like ordinary records. However, because of the restrictions on virtual records, they can also be implemented in a type-erased setting by simply ignoring all virtual record projections and replacing each virtual record by any one of the record's components. This makes sense because all of the components of one virtual record are merely differently type-annotated versions of the same untyped term.

Example 2.3. Figure 3 shows the syntax tree of the λ^{CIL}-term that corresponds to the untyped term $\hat{M}$ with type $\wedge\{i = \hat{\tau},\ j = \hat{\sigma}\}$ from Example 2.2. Note that the tree contains three duplicates of the subterm $(\lambda w.\ w\ x\ y)$ and two duplicates of $\check{M}$ itself.

□

The important drawback of λ^{CIL} that we are concerned about in this paper is that duplicate terms complicate local program transformations like β-reduction. In λ^{CIL}, a β-reduction of a subterm must be repeated in all its duplicate subterms in order to preserve well-typedness. For this reason, in λ^{CIL} the local transformation of β-reduction is simulated by a global transformation involving simultaneous replacements in all of the positions identified by a *parallel context* (for one definition of this concept see [WDMT02]). In other words, β-reduction on typed terms is not *contextual*, i.e., in λ^{CIL}, if $M \rightarrow_\beta N$ holds this does not imply that $C[M] \rightarrow_\beta C[N]$ holds for an arbitrary one-hole context C.

2.4 The Example in Our New System λ^{B}

In this paper, we propose the system λ^{B} of branching types — a type system with explicit type annotations that is equivalent in some sense to intersection types but does not need duplicate representatives in the typed terms of subterms of the untyped terms. The type annotations in λ^{B} contain the same information as the type annotations in λ^{CIL}. Thus, λ^{B}

(ax)
$$\overline{\hat{E} \vdash^{\mathsf{c}} x^{\hat{E}(x)} : \hat{E}(x)}$$

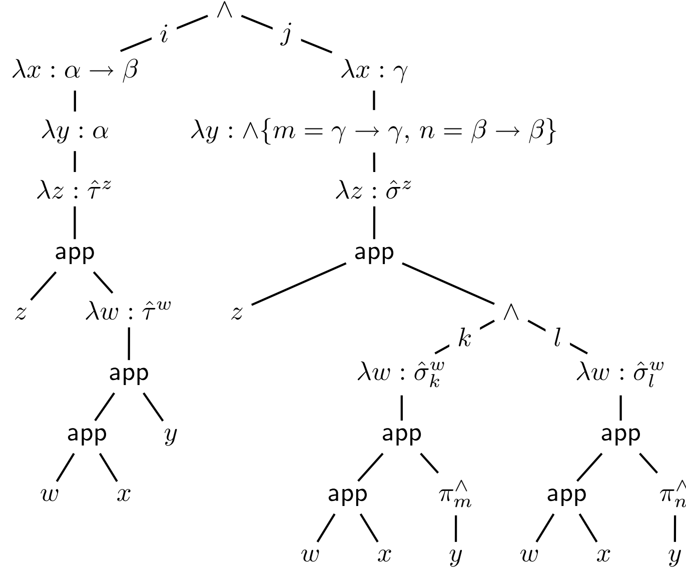
($\to_{\mathsf{i}}$)
$$\frac{\hat{E}[x \mapsto \hat{\sigma}] \vdash^{\mathsf{c}} \breve{M} : \hat{\tau}}{\hat{E} \vdash^{\mathsf{c}} \lambda x^{\hat{\sigma}}.\breve{M} : \hat{\sigma} \to \hat{\tau}}$$

($\to_{\mathsf{e}}$)
$$\frac{\hat{E} \vdash^{\mathsf{c}} \breve{M} : \hat{\sigma} \to \hat{\tau}; \quad \hat{E} \vdash^{\mathsf{c}} \breve{N} : \hat{\sigma}}{\hat{E} \vdash^{\mathsf{c}} \breve{M}\ \breve{N} : \hat{\tau}}$$

($\wedge_{\mathsf{i}}$)
$$\frac{\hat{E} \vdash^{\mathsf{c}} \breve{M}_i : \hat{\tau}_i \text{ for all } i \in I}{\hat{E} \vdash^{\mathsf{c}} \wedge\{i = \breve{M}_i\}^I : \wedge\{i = \hat{\tau}_i\}^I} \quad \text{if } |\breve{M}_i| = |\breve{M}_j| \neq \bot \text{ for all } i, j \in I$$

($\wedge_{\mathsf{e}}$)
$$\frac{\hat{E} \vdash^{\mathsf{c}} \breve{M} : \wedge\{i = \hat{\tau}_i\}^I}{\hat{E} \vdash^{\mathsf{c}} \pi^{\wedge}_j\ \breve{M} : \hat{\tau}_j} \quad \text{if } j \in I$$

Figure 2: λ^{CIL} — typing rules



To avoid superscripts on superscripts in this figure and also in Figure 4, we have written each $\lambda x^{\hat{\tau}}$ instead as $\lambda x : \hat{\tau}$. For brevity, we have omitted redundant type annotations on non-binding variable occurrences.

Figure 3: λ^{CIL} — a syntax tree

is suitable for the same kinds of uses as λ^{CIL}, such as customizing functions in multiple ways for different uses based on type annotations. On the other hand, unlike in λ^{CIL}, β-reduction in λ^{B} is contextual.

The basic idea for λ^{B} is to refine the language of types, so that it allows to express that a single term has several types in several different branches of a λ^{I}-derivation. The formal definition of λ^{B} is in Section 7. Here, we give an introductory flavor by re-expressing Examples 2.2 and 2.3 in λ^{B}.

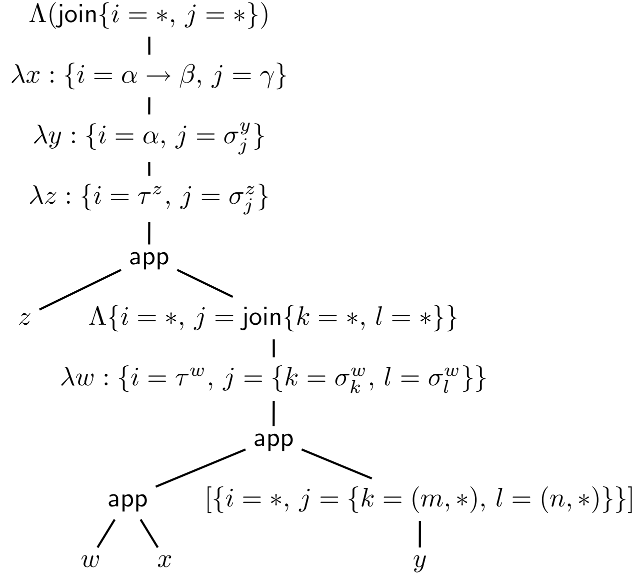


Figure 4: λ^{B} — a syntax tree

Example 2.4. In λ^{B}, the term $\hat{M}$ from Example 2.2 can be annotated to have the type ρ defined below, which corresponds to the type $\wedge\{i = \tau,\ j = \sigma\}$ from Example 2.2. The types $\tau, \tau^z, \tau^w, \sigma^w_k, \sigma^w_l$ are the same as in Example 2.2, but are repeated here for the reader's convenience.

$$\begin{array}{rcl}
\rho & = & \forall(\mathsf{join}\{i = *,\ j = *\}).\,\{i = \tau,\ j = \sigma^j\} \\
\tau & = & (\alpha \to \beta) \to \alpha \to \tau^z \to \beta \\
\tau^z & = & (\tau^w \to \beta) \to \beta \\
\tau^w & = & (\alpha \to \beta) \to \alpha \to \beta \\
\sigma^j & = & \gamma \to \sigma^y_j \to \sigma^z_j \to \beta \\
\sigma^y_j & = & \forall(\mathsf{join}\{m = *,\ n = *\}).\,\{m = \gamma \to \gamma,\ n = \beta \to \beta\} \\
\sigma^z_j & = & (\forall(\mathsf{join}\{k = *,\ l = *\}).\,\{k = \sigma^w_k \to \beta,\ l = \sigma^w_l \to \beta\}) \to \beta \\
\sigma^w_k & = & \gamma \to (\gamma \to \gamma) \to \beta \\
\sigma^w_l & = & \gamma \to (\beta \to \beta) \to \beta
\end{array}$$

The syntax tree of the corresponding λ^{B}-term is shown in Figure 4.

The reader should first look at type annotations on variables. Note, for instance, that the type annotation on the binding occurrence of x is of the shape $\{i = \cdots,\ j = \cdots\}$. This is so because in λ^{I} the derivation for the term $\hat{M} = (\lambda x.\lambda y.\lambda z.\, z\ (\lambda w.\, w\ x\ y))$ has two subderivations for the entire term. The i branch of x's type $\{i = \alpha \to \beta,\ j = \gamma\}$ holds x's type in the i subderivation of the corresponding λ^{I}-type-derivation, while the j branch holds

x's type in the j subderivation. The type annotation on the binding occurrence of w has the shape $\{i = \cdots, j = \{k = \cdots, l = \cdots\}\}$. This is so because the j subderivation of the corresponding λ^{I}-derivation has two different subderivations for $(\lambda w.\, w\ x\ y)$. The type $\{i = \tau^w, j = \{k = \sigma_k^w, l = \sigma_l^w\}\}$ that annotates w's binding occurrence may be interpreted as follows:

> w has type τ^w in the i subderivation, type σ_k^w in the jk subderivation and type σ_l^w in the jl subderivation of the corresponding λ^{I}-derivation.

Next, the reader can look at the two nodes of the form $\Lambda\{\cdots\}$. These nodes correspond to uses of the intersection introduction rule in the λ^{I}-derivation. Consider the node $\Lambda\{i = *, j = \mathsf{join}\{k = *, l = *\}\}$. This node may be interpreted as follows:

> Join the jk-subderivation and the jl-subderivations using λ^{I}'s intersection introduction, and do nothing in the i-subderivation.

Here again $*$ corresponds to the subderivations of the corresponding λ^{I} derivation that do not contain any nested uses of intersection introduction.

Finally, the single node in Figure 4 that corresponds to the two uses of λ^{I}'s intersection elimination rule is $[\{i = *, j = \{k = (m, *), l = (n, *)\}\}]$. This node may be interpreted as follows:

> In the jk-subderivation select the m-component of an intersection type of the shape $\wedge\{\cdots, m = \cdots\}$, in the jl-subderivation select the n-component of an intersection type of the shape $\wedge\{\cdots, n = \cdots\}$, and in the i-subderivation do nothing.

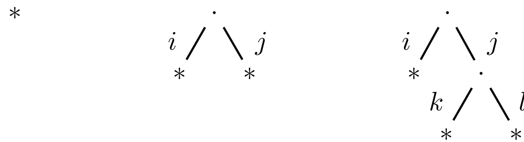
□

3 Discussion of Features of λ^{B}

In this section, the example from the previous section will be used to illustrate the main features of λ^{B}.

Branching Shapes. Every type and typing judgment of λ^{B} has a *branching shape*. In λ^{B}, branching shapes are essentially trees with finite height and branching whose edges are labeled by elements of the fixed label set $\mathcal{I}$. These trees correspond in a certain way to the branching shapes of type derivations in λ^{I} and, similarly, to the branching shapes of terms in λ^{CIL}.

The branching shapes used in the λ^{B} example correspond to the shape of the example λ^{CIL} syntax tree, which can be seen to have 2 uses of $(\wedge_{\mathsf{i}})$, one at the root with branches labeled i (left) and j (right), and another inside the j (right) child of the root with branches labeled k (left) and l (right). The branching shapes that are used for typing judgments for the λ^{B} example are, from the root to the leaves, $*$, $\{i = *, j = *\}$, and $\{i = *, j = \{k = *, l = *\}\}$. We always use $*$ for the leaves of branching shapes. These branching shapes can be viewed as the following three edge-labeled trees:



There is one additional branching shape that is relevant to the example. The binding type of y, which is $\{i = \alpha, j = \sigma_j^y\}$, is equivalent (the equivalence is discussed further below)

to a type of the shape $\forall P.\rho'$ where the branching shape of ρ' is $\{i = *,\ j = \{m = *, n = *\}\}$:

```
        .
     i / \ j
      *   .
       m / \ n
        *   *
```

The $\{m = *, n = *\}$ portion of this shape corresponds to the intersection type $\wedge\{m = \gamma \to \gamma,\ n = \beta \to \beta\}$ assigned to y in one of the subderivations of the λ^{I}-derivation for $\hat{M}$. There are no m and n subderivations inside the derivation for $\hat{M}$. Instead when $\hat{M}$ is used in a larger context that applies it to arguments, an argument that gets bound to y must have m and n subderivations.

Branching Types. Consider the λ^{CIL} tree in Figure 3 and the corresponding λ^{B} tree in Figure 4. The λ^{CIL} tree can be divided into the root, the portion inside the i branch (left branch from the root), and the portion inside the j branch (right branch from the root. The j branch can be further subdivided into the portion not below the use of intersection introduction (which is marked with $\wedge$), the portion inside the k sub-branch (to the left of the $\wedge$), and the portion inside the l sub-branch.

The λ^{B} tree is a straightforward transformation of the λ^{CIL} tree. All of the types in the i and j branches of the λ^{CIL} tree have been placed inside the label i and j respectively in the λ^{B} types. Similarly, the types in the k and l subbranches of the inner $(\wedge_{\mathsf{i}})$ rule in the λ^{CIL} tree are not only inside the label j in the λ^{B} types, but also inside the label k and l respectively.

For example, the binding type of w in the λ^{B} tree is the *branching type*

$$\{i = \tau^w,\ j = \{k = \sigma^w_k,\ l = \sigma^w_l\}\}$$

which has the branching shape $\{i = *,\ j = \{k = *,\ l = *\}\}$. This corresponds to the fact that in the λ^{CIL} derivation the binding of w is duplicated 3 times and occurs in the branches we have named i, jk, and jl, where it binds respectively the types τ^w, σ^w_k, and σ^w_l.

Type Equivalence. In λ^{B}, there is a rewriting relation on types named $\to_{\mathsf{ty}}$ that puts types into a canonical form, and there is an equivalence relation on types named $\leftrightarrow_{\mathsf{ty}}$ that is the smallest equivalence relation containing $\to_{\mathsf{ty}}$. (Details can be found in Definition 7.4.) There is a λ^{B} typing rule that allows replacing a derived result type by any equivalent type (using $\leftrightarrow_{\mathsf{ty}}$). Of the rules defining $\to_{\mathsf{ty}}$, a particularly important rule is the following one, because it effectively allows using the usual typing rules for λ-calculus abstraction and application in combination with branching types:

$$(\{i = \sigma_i\}^I \to \{i = \tau_i\}^I) \to_{\mathsf{ty}} \{i = \sigma_i \to \tau_i\}^I$$

To see why this rule is necessary, consider the subterm $\mathsf{app}(w, x)$ in the tree in Figure 4. Because the variable w is used as a function, one would expect that it has a function type, i.e., a type of the form $\rho' \to \rho''$. However, at its binding site, w is associated with the type $\{i = \tau^w,\ j = \{k = \sigma^w_k,\ l = \sigma^w_l\}\}$, which is not written in the usual form of a function type. Using the type equivalences, one can show that this type is equivalent to $\rho^w_{\mathsf{d}} \to \rho^w_{\mathsf{c}}$, where ρ^w_{d} and ρ^w_{c} are defined as follows:

$$\begin{aligned}
\rho^w_{\mathsf{d}} &= \{i = \alpha \to \beta,\ j = \{k = \gamma,\ l = \gamma\}\} \\
\rho^w_{\mathsf{c}} &= \{i = \alpha \to \beta,\ j = \{k = (\gamma \to \gamma) \to \beta,\ l = (\beta \to \beta) \to \beta\}\}
\end{aligned}$$

The derivation of this equivalence using the rules proceeds as follows:

$$\begin{array}{ll}
 & \{i = \tau^w,\ j = \{k = \sigma_k^w,\ l = \sigma_l^w\}\} \\
= & \{i = \tau^w,\ j = \{k = \gamma \to (\gamma \to \gamma) \to \beta,\ l = \gamma \to (\beta \to \beta) \to \beta\}\} \\
\leftarrow_{\mathsf{ty}} & \{i = \tau^w,\ j = \{k = \gamma, l = \gamma\} \to \{k = (\gamma \to \gamma) \to \beta,\ l = (\beta \to \beta) \to \beta\}\} \\
= & \{i = (\alpha \to \beta) \to \alpha \to \beta,\ j = \{k = \gamma, l = \gamma\} \to \{k = (\gamma \to \gamma) \to \beta,\ l = (\beta \to \beta) \to \beta\}\} \\
\leftarrow_{\mathsf{ty}} & \quad \{i = \alpha \to \beta,\ j = \{k = \gamma, l = \gamma\}\} \\
 & \to \{i = \alpha \to \beta,\ j = \{k = (\gamma \to \gamma) \to \beta,\ l = (\beta \to \beta) \to \beta\}\} \\
= & \rho_{\mathsf{d}}^w \to \rho_{\mathsf{c}}^w
\end{array}$$

Hence, although w's type is not in the syntactic form of a function type, it is equivalent to a function type and can thus be used as one.

Type Selection. In λ^{B}, *type selection* is a feature that replaces the $(\wedge_{\mathsf{i}})$ and $(\wedge_{\mathsf{e}})$ rules of λ^{CIL}. At the level of terms, type selection involves *abstraction* over *parameters*, written $(\Lambda P.\Box)$, and *application* to *arguments*, written $\Box\,[A]$. At the level of types, type selection involves quantification over type selection parameters, written $\forall P.\Box$, as well as 2 additional type rewriting rules. In λ^{B}, quantification over type selection parameters replaces the intersection types of λ^{CIL}.

The term-level type selection abstraction $(\Lambda P.\Box)$ corresponds to some number of uses of the $(\wedge_{\mathsf{i}})$ rule of λ^{CIL}. Here, P is a *type selection parameter* — a pattern that records the current branching shape of both the λ^{B} term that will be placed in the hole and of the resulting term. If M has type τ and $N = \Lambda P.M$ is well typed, then N will have the type $\forall P.\tau$.

Similarly, the term-level type selection application $\Box\,[A]$ corresponds to some number of uses of λ^{CIL}'s $(\wedge_{\mathsf{e}})$ rule. Here, A is a *type selection argument*. For example, the type selection argument $\{i = *,\ j = \{k = (m, *),\ l = (n, *)\}\}$ indicates that in the corresponding λ^{CIL}-term nothing at all is done in the i-branch, the m-component of a virtual record is selected in the jk-branch, and the n-component is selected in the jl-branch. Here the $*$ is used as an indicator of "no further action" at the end of each chain of branch selections.

Each intersection type $\rho_1 \wedge \rho_2$ in the λ^{CIL} example has a corresponding type of the shape $\forall(\mathsf{join}\{f_1 = *, f_2 = *\}).\{f_1 = \rho_1', f_2 = \rho_2'\}$ in the λ^{B} example. A type of the shape $\forall P.\rho'$ has a *type selection parameter* P, which is a pattern indicating what possible *type selection arguments* are valid to supply. Each parameter P has 2 branching shapes, its *lower branching shape* $\lfloor P \rfloor$ and its *upper branching shape* $\lceil P \rceil$. The branching shape of a type $\forall P.\rho'$ is $\lfloor P \rfloor$ and the branching shape of ρ' must be $\lceil P \rceil$.

Type Expansion. In the subterm $\mathsf{app}(w, x)$, the type of variable x should coincide with the domain type of w's function type, i.e., with

$$\{i = \alpha \to \beta,\ j = \{k = \gamma,\ l = \gamma\}\}$$

However, at its binding site, x is associated with the type

$$\{i = \alpha \to \beta,\ j = \gamma\}$$

These two types differ because the type selection abstraction

$$\Lambda\,\{i = *,\ j = \mathsf{join}\{k = *,\ l = *\}\}$$

intervenes between the two occurrences of x. As described above, this particular type selection abstraction corresponds to the inner use of the intersection introduction typing rule in the corresponding λ^{I} derivation and the inner virtual record constructor in the corresponding λ^{CIL} term in Figure 3. So x has different types at different occurrences. The type of a free variable inside a $(\Lambda P.\Box)$ form (where P is some type selection parameter) is *expanded*

relative to its type outside the same form. (Type expansion is defined in Section 7.2.1.) In the example, x's type is $\{i = \alpha \to \beta,\, j = \gamma\}$ at its binding site but below the syntax tree node $\Lambda\{i = *,\, j = \mathsf{join}\{k = *,\, l = *\}\}$ the type of x is $\{i = \alpha \to \beta,\, j = \{k = \gamma,\, l = \gamma\}\}$. Thus, in the subterm $\mathsf{app}(w, x)$, x's type matches w's domain type as desired.

Type Equivalence for Type Selection. The additional type rewriting rules that are needed to make type selection work are the following:

$$\forall *.\tau \to_{\mathsf{ty}} \tau \qquad\qquad \forall\,\{i = P_i\}^I\,.\,\{i = \tau_i\}^I \to_{\mathsf{ty}} \{i = \forall P_i.\tau_i\}^I$$

To illustrate their use, we explain why the application of y to its type selection argument is well typed in the example. The type of y at its binding site is $\{i = \alpha,\, j = \sigma^y_j\}$ and has branching shape $\{i = *,\, j = *\}$. Because the leaf occurrence of y must have branching shape $\{i = *,\, j = \{k = *,\, l = *\}\}$, the type at that location is expanded by the typing rules to be $\{i = \alpha,\, j = \{k = \sigma^y_j,\, l = \sigma^y_j\}\}$. For the sake of the example, we now use some additional names to abbreviate portions of the types:

$$\begin{aligned} P_{mn} &= \mathsf{join}\{m = *,\, n = *\} \\ \sigma^y_{j2} &= \{m = \gamma \to \gamma,\, n = \beta \to \beta\} \end{aligned}$$

Thus, $\sigma^y_j = \forall P_{mn}.\sigma^y_{j2}$. Type equivalences can now be applied to the type as follows to lift the occurrences of $\forall P$ to outermost position:

$$\begin{aligned} & \{i = \alpha,\, j = \{k = \sigma^y_j,\, l = \sigma^y_j\}\} \\ \leftrightarrow_{\mathsf{ty}}\; & \{i = (\forall *.\alpha),\, j = \forall\{k = P_{mn},\, l = P_{mn}\}.\{k = \sigma^y_{j2},\, l = \sigma^y_{j2}\}\} \\ \leftrightarrow_{\mathsf{ty}}\; & \forall\{i = *,\, j = \{k = P_{mn},\, l = P_{mn}\}\}.\{i = \alpha,\, j = \{k = \sigma^y_{j2},\, l = \sigma^y_{j2}\}\} \end{aligned}$$

The final type is the type derived (but not shown) in figure 4 for the occurrence of y. The type selection parameter in this type is $\{i = *,\, j = \{k = P_{mn},\, l = P_{mn}\}\}$ which effectively says, "in the i branch, no type selection argument can be supplied and in the jk and jl branches, a choice between m and n can be supplied". The type selection argument $\{i = *, j = \{k = (m, *), l = (n, *)\}\}$ in fact does just this; it supplies no choice in the i branch, a choice of m in the jk branch, and a choice of n in the jl branch. The $*$ in $(m, *)$ means that after the choice of m is supplied, no further choices are supplied.

Term Reduction. The term rewriting rules (see Definition 10.5) manipulate and simplify the type annotations as needed. As an example, consider the term

$$M = (\Lambda P_1.\Lambda P_2.\lambda x^\tau.x)\,[A]\,[B]\,y$$

where the type annotation parts of the term are:

$$\begin{aligned} P_1 &= \{m = \mathsf{join}\{j = *,\, k = *\},\, n = *\}, \\ P_2 &= \{m = *,\, n = \mathsf{join}\{h = *,\, l = *\}\}, \\ A &= \{m = *,\, n = (h, *)\}, \\ B &= \{m = (j, *),\, n = *\}, \\ \tau &= \{m = \{j = \alpha_1,\, k = \alpha_2\},\, n = \{h = \beta_1,\, l = \beta_2\}\} \end{aligned}$$

This is a well typed λ^{B}-term, if the free variable y is assumed to have type $\{m = \alpha_1,\, n = \beta_1\}$. Erasing the type annotations from M results in the untyped term $(\lambda x.x)\,y$, which β-reduces to y. Our term rewriting rules (see Definition 10.5) can simulate this single β-reduction step for untyped terms, although the simulation requires more than one typed rewriting step. First, a sequence of rewriting steps cancels the type selection parameters and arguments that block an immediate β-reduction; then, a β-reduction step results in y.

Here are the rewriting steps in detail for the example term M. The first step is

$$M = (\Lambda P_1.\Lambda P_2.\lambda x^\tau.x)\,[A]\,[B]\,y \to_{\mathsf{b}} (\Lambda P_1.(\Lambda P_2.\lambda x^\tau.x)\,[A])\,[B]\,y$$

using the (Select) rule, where in this particular case P_1 and A pass through each other without interacting because P_1 only does something interesting in the m branch and A only does something interesting in the n branch. The next step is

$$(\Lambda P_1.(\Lambda P_2.\lambda x^{\tau}.x)\,[A])\,[B]\,y \rightarrow_{\mathsf{b}} (\Lambda P_1.\Lambda P_2'.(\lambda x^{\tau'}.x)[A'])\,[B]\,y$$

where

$$P_2' = A' = \{m = *,\, n = *\} \text{ and}$$
$$\tau' = \{m = \{j = \alpha_1,\, k = \alpha_2\},\, n = \beta_1\},$$

again using the (Select) rule. In this case, P_2 and A interact because they both do something interesting in the n branch. In particular, P_2 offers a choice of h and l sub-branches of the n branch and A chooses the h sub-branch; this explains the transformation from the type τ to τ'. In this particular case, the residual P_2' and A' left over after the interaction are both *trivial* and will subsequently evaporate. In the more general case, an interaction between a parameter P and an argument A can use up part of the parameter and argument while leaving non-trivial parts behind. The next two steps are

$$(\Lambda P_1.\Lambda P_2'.(\lambda x^{\tau'}.x)[A'])\,[B]\,y \rightarrow_{\mathsf{b}} (\Lambda P_1.\Lambda P_2'.\lambda x^{\tau'}.x)\,[B]\,y \rightarrow_{\mathsf{b}} (\Lambda P_1.\lambda x^{\tau'}.x)\,[B]\,y$$

first removing the trivial A' by the $(*_A)$ rule and then removing the trivial P_2' by the $(*_P)$ rule. The $(*_P)$ and $(*_A)$ rules simply remove *trivial* parameters and arguments respectively, where a parameter or arguments is trivial iff it is indistinguishable from a branching shape. Trivial parameters and arguments effectively do nothing to the types, so it makes sense to simply remove them. The next step is

$$(\Lambda P_1.\lambda x^{\tau'}.x)\,[B]\,y \rightarrow_{\mathsf{b}} (\Lambda P_1'.(\lambda x^{\tau''}.x)\,[B'])\,y$$

where

$$P_1' = B' = \{m = *,\, n = *\} \text{ and}$$
$$\tau'' = \{m = \alpha_1,\, n = \beta_1\}$$

using the (Select) rule. In this case, the parameter P_1 offers a choice of j and k sub-branches of the m branch and the argument B selects the j sub-branch. This transforms τ' into τ'', which is the environment type of y. The next two steps are

$$(\Lambda P_1'.(\lambda x^{\tau''}.x)\,[B'])\,y \rightarrow_{\mathsf{b}} (\Lambda P_1'.\lambda x^{\tau''}.x)\,y \rightarrow_{\mathsf{b}} (\lambda x^{\tau''}.x)\,y$$

removing the trivial B' and P_1' using the $(*_A)$ and $(*_P)$ rules. Finally, the (β) rule is used:

$$(\lambda x^{\tau''}.x)\,y \rightarrow_{\mathsf{b}} y$$

4 Comparison to Recent Related Work

Ronchi Della Rocca and Roversi have a system called Intersection Logic (IL) [RDRR01] which is similar to λ^{B}, but has nothing corresponding to our explicitly typed terms. IL has a meta-level operation corresponding to our equivalence for function types. IL has nothing corresponding to our other type equivalences, because IL does not group parallel occurrences of its equivalent of type selection parameters and arguments, but instead works with equivalence classes of derivations modulo permutations of what we call *individual* type selection parameters and arguments. We expect that the use of these equivalence classes will cause great difficulty with the proofs for IL. It would be interesting to see complete proofs of the properties of IL. A proof-term-labeled version of IL is presented, but the proof terms are pure λ-terms and thus the proof terms do not represent entire derivations.

Capitani, Loreti, and Venneri have designed a similar system called HL (Hyperformulae Logic) [CLV01]. HL is quite similar to IL, although it seems overall to have a less complicated

presentation. HL has nothing corresponding to our equivalences on types. The set of properties proved for HL in [CLV01] is not exactly the same as the set of properties proved for IL in [RDRR01], e.g., there is no attempt to directly prove any result related to reduction of HL proofs as there is for IL, although this could be obtained indirectly via their proofs of equivalence with traditional systems with intersection types. HL is reported in [RDRR01] to have a typed version of an untyped calculus like that in [Kfo00], but in fact there is no significant connection between [CLV01] and [Kfo00] and there is no explicitly typed calculus associated with HL.[1]

As for λ^{B}, for both IL and HL there are proofs of equivalence with more traditional systems with intersection types. These proofs show that the proof-term-annotated versions of IL and HL can type the same sets of pure untyped λ-terms as a traditional system with intersection types. The correspondence we show for λ^{B} involves a notion of type erasure while the correspondences for IL and HL do not as their proof terms are pure non-type-annotated λ-terms.

In recent work [LRDR05], Liquori and Ronchi Della Rocca present a system $\Lambda\mathcal{P}$ that is isomorphic to the fragment of λ^{CIL} from Section 2.3. $\Lambda\mathcal{P}$-terms are called "tree stores". The authors present a second system $\Lambda^{\mathrm{t}}_{\wedge}$, whose typing judgment associates with untyped λ-terms both a type and a tree store (λ^{CIL}-term). β-reduction reduces untyped λ-terms and tree stores (λ^{CIL}-terms) in parallel. Unfortunately, [LRDR05] omits the definition of reduction context for this setting. We assume that, like λ^{CIL} [WDMT02], the full definition of β-reduction uses parallel reduction contexts for tree stores, because other notions of reduction context seem to either break subject reduction or prohibit certain β-reductions that are possible for untyped terms. Liquori and Ronchi Della Rocca's system is, thus, more closely related to λ^{CIL} than to λ^{B} and, in particular, does not attempt to avoid parallel reduction contexts like we do.

5 Conclusion

In this paper, we present λ^{B}, the first explicitly typed calculus with the power of intersection types which is Church-style, i.e., typed terms contain explicit type annotations and do not have multiple disjoint subterms corresponding to single subterms in the corresponding untyped term. Branching types are used in λ^{B} to represent the effect of what is handled with simultaneous derivations in systems with intersection types. We prove for λ^{B} subject reduction, and soundness and completeness of explicitly typed reduction w.r.t. β-reduction on the corresponding untyped λ-terms. Moreover, we formally relate λ^{B} to a traditional intersection type system where terms do not have explicit type annotations. The main benefit of λ^{B} will be to make it easier to use technology (both theories and software) already developed for the λ-calculus on explicitly typed terms in a type system having the power and flexibility of intersection types. Due to the experimental performance measurements reported in [DWM+01b], we do not expect a substantial size benefit in practice from λ^{B} over λ^{CIL}. In the area of logic, λ^{B} terms may be useful as explicitly typed realizers of the so-called *strong conjunction*, but we are not currently planning on investigating this ourselves.

6 Acknowledgments

Assaf Kfoury started this all off by helping to put the first author in the situation in December of 1994 where he needed an explicitly typed system with intersection types. At that time, the first author developed two ideas; one became λ^{CIL} and the other λ^{B}. The first author then misled the compiler implementers of the Church Project (Allyn Dimock, Glenn Holloway, Bob Muller, Lyn Turbak, Ian Westmacott, et al.) into thinking λ^{CIL} is

[1] It seems there may have been one in an unpublished version of the paper.

good enough and spent a number of years working with them to learn why this is not true and a system like λ^{B} is needed instead. All off the members of the Church Project as well as Paweł Urzyczyn very helpfully listened to and commented on earlier versions of these ideas. We would also like to thank the anonymous referees of versions of this paper for their very helpful comments.

Part II
Technical Presentation

This part presents formal definitions, lemmas, and theorems as well as a number of examples. The reader interested in the motivations and implications of our new system λ^{B} will find them in part I.

6.1 Partial Functions

In this paragraph, we fix some definitions and notational conventions concerning partial functions: If X is a set not containing an element by the name $\bot$, then $\mathbf{X}_\bot$ denotes the partially ordered set $(X_\bot, \leq)$, where

$$X_\bot \stackrel{\text{def}}{=\!=} X \cup \{\bot\}\ ; \qquad (x \leq y) \stackrel{\text{def}}{\iff} (x = \bot)$$

A function from $X_\bot$ to $Y_\bot$ is called *strict* if it maps $\bot$ to $\bot$. In this article, *partial functions* from a set X to a set Y are defined as strict, total functions from $X_\bot$ to $Y_\bot$. Suppose that f is a partial function from X to Y, $x \in X$ and $y \in Y$. We say that $f(x)$ is *undefined* if $f(x) = \bot$. Otherwise, we say that $f(x)$ is *defined*. The domain of f is the set $\{\, x \mid f(x) \text{ is defined} \,\}$ and is denoted by $\mathsf{dom}(f)$. The range of f is the set $\{\, f(x) \mid x \in \mathsf{dom}(f) \,\}$ and is denoted by $\mathsf{ran}(f)$. The expression $f[x \mapsto y]$ denotes the partial function $\{\, (x', y) \in f \mid x' \neq x \,\} \cup \{(x, y)\}$. A *finite function* from X to Y is a partial function from X to Y that has a finite domain. Partial functions are *ordered pointwise*. That is, if f, g are partial functions from X to Y, then $(f \leq g)$ iff $(f(x) \leq g(x))$ for all $x \in X$. We will often define partial functions inductively by sets of equations. Such inductive definitions define the least partial function (with respect to the pointwise ordering) that satisfies the given equations. In this paper, tuple formation is strict, i.e., $(x_1, \ldots, x_n) = \bot$ iff $x_i = \bot$ for some i in $\{1, \ldots, n\}$. For instance, if f is a ternary partial function, g a unary partial function and $g(y) = \bot$, then $f(g(x), g(y), z) = \bot$.

Given a binary relation $\mathcal{R}$ on a set X, we lift it to a binary relation $\mathcal{R}^\bot$ on $X_\bot$ as follows:

$$(x_1\ \mathcal{R}^\bot\ x_2) \stackrel{\text{def}}{\iff} (x_1 = x_2 = \bot) \text{ or } (x_1\ \mathcal{R}\ x_2)$$

We will make use of the following facts:

- If $\mathcal{R}$ is an equivalence relation on X, then $\mathcal{R}^\bot$ is an equivalence relation on $X_\bot$.
- If $\mathcal{R}$ is an equivalence relation on X, $(x\ \mathcal{R}^\bot\ y)$, and $(y\ \mathcal{R}\ z)$, then $(x\ \mathcal{R}\ z)$. (The omissions of the superscripts are intentional.)

7 The Branching Type System λ^{B}

7.1 Types and Branching shapes

7.1.1 Branching Shapes

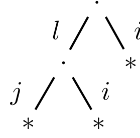
The set BShape of *branching shapes* is defined by the following pseudo-grammar:

$$\kappa \in \mathsf{BShape} \ ::= \ * \mid \{i = \kappa_i\}^I$$

One may view branching shapes as edge-labeled trees. For example, the branching shape

$$\{l = \{j = *,\, i = *\},\, i = *\}$$

may be viewed as the following tree:



We define a partial order on branching shapes, inductively by the following rules:

$$\frac{}{* \le \kappa} \qquad \frac{(\kappa_i \le \kappa'_i) \text{ for all } i \in I}{\{i = \kappa_i\}^I \le \{i = \kappa'_i\}^I}$$

Thus, $(\kappa \le \kappa')$ iff the tree κ is a prefix of the tree κ'.

Lemma 7.1. *The relation $\le$ is a partial order on the set of branching shapes.* □

7.1.2 Types

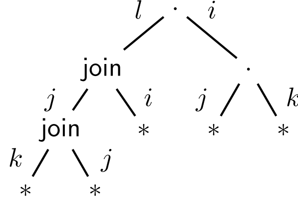
The sets Parameter of *type selection parameters* and IndParameter of *individual type selection parameters* are defined by the following pseudo-grammar:

$$\begin{array}{rcl} P \in \mathsf{Parameter} & ::= & \bar{P} \;\mid\; \{i = P_i\}^I \\ \bar{P} \in \mathsf{IndParameter} & ::= & * \;\mid\; \mathsf{join}\{i = \bar{P}_i\}^I \end{array}$$

Like branching shapes, type selection parameters may be viewed as edge-labeled trees. However, in type selection parameters some of the internal nodes are labeled by the token join. For example, the type selection parameter

$$\{l = \mathsf{join}\{j = \mathsf{join}\{k = *,\, j = *\},\, i = *\},\, i = \{j = *,\, k = *\}\}$$

may be viewed as the following tree:

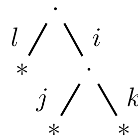


Note that if an internal node is labeled by join, then all the internal nodes below that node must also be labeled by join. Individual type selection parameters are those type selection parameters where *all* internal nodes are labeled by join.

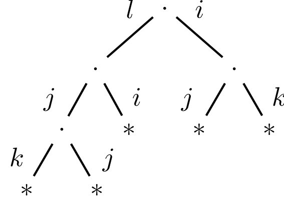
Given a parameter P, let P's *lower branching shape* $\lfloor P \rfloor$ and *upper branching shape* $\lceil P \rceil$ be defined as follows:

$$\begin{array}{lll} \lfloor * \rfloor = *, & \lfloor \mathsf{join}\{i = \bar{P}_i\}^I \rfloor = *, & \lfloor \{i = P_i\}^I \rfloor = \{i = \lfloor P_i \rfloor\}^I \\ \lceil * \rceil = *, & \lceil \mathsf{join}\{i = \bar{P}_i\}^I \rceil = \{i = \lceil \bar{P}_i \rceil\}^I, & \lceil \{i = P_i\}^I \rceil = \{i = \lceil P_i \rceil\}^I \end{array}$$

If P is the type selection parameter from above, then the following tree depicts its lower branching shape



and the following one its upper branching shape



Lemma 7.2. $\lfloor P \rfloor \leq \lceil P \rceil$. □

The set Ty of types is defined by the following pseudo-grammar:

$$\sigma, \tau \in \mathsf{Ty} \quad ::= \quad \alpha \;\mid\; \sigma \to \tau \;\mid\; \{i = \tau_i\}^I \;\mid\; \forall P.\tau$$

A relation assigning branching shapes to types is defined inductively by the following rules:

$$\frac{}{\alpha : *} \qquad \frac{\sigma : \kappa; \quad \tau : \kappa}{\sigma \to \tau : \kappa} \qquad \frac{\tau_i : \kappa_i \text{ for all } i \in I}{\{i = \tau_i\}^I : \{i = \kappa_i\}^I} \qquad \frac{\tau : \lceil P \rceil}{\forall P.\tau : \lfloor P \rfloor}$$

Note that not every type has a branching shape. On the other hand, every type has at most one branching shape. Let a type τ be called *well formed* iff there is a branching shape κ such that $(\tau : \kappa)$. Let a well formed type be called *individual* iff its branching shape is $*$. The set of all individual types is denoted by IndTy. Let a well formed type be called *branching* iff it is not individual.

Note also that type variables are always of branching shape $*$. We considered the idea of allowing type variables of higher branching shapes. However, this would have complicated the system by requiring type environments (defined later in Section 7.4) to contain branching shapes of type variables in addition to types of term variables and to be sequences rather than finite maps. Furthermore, the effect of a higher branching shape type variable can be simulated, e.g., a (hypothetical) type variable α of kind $\{i = *, j = *\}$ can be effectively simulated by the λ^{B} type $\{i = \alpha_i, j = \alpha_j\}$.

Individual types directly correspond to intersection types, the join corresponding to $\wedge$. On the other hand, multiple branches in a type's branching shape correspond, in a certain sense, to multiple type derivations for a single term in an intersection type system. The precise relation between our branching type system and an intersection type system will be described in Section 9. To get an idea of the relation, consider the following example:

Example 7.3. The untyped term $\lambda x.x$ can be given many types in λ^{I}, for example $(\alpha \to \alpha)$ and $((\alpha \to \alpha) \to \alpha \to \alpha)$. As a result, in λ^{B} this term (or, more precisely, a corresponding explicitly typed term) can be given the following branching type:

$$\{i = \alpha \to \alpha,\; j = (\alpha \to \alpha) \to \alpha \to \alpha\}$$

Joining the two components of this branching type, results in the following well formed individual type:

$$\forall \mathsf{join}\,\{i = *, j = *\}\,.\,\{i = \alpha \to \alpha,\; j = (\alpha \to \alpha) \to \alpha \to \alpha\}$$

This type is also a type of $\lambda x.x$ in λ^{B}, corresponding to the following intersection type in λ^{I}:

$$\wedge\{i = \alpha \to \alpha,\; j = (\alpha \to \alpha) \to \alpha \to \alpha\}$$ □

7.1.3 Type Equivalence

In order to be able to treat certain types as having essentially the same meaning, we define an equivalence relation on the set of types as follows. Let a binary relation $\mathcal{R} \subseteq \mathsf{Ty} \times \mathsf{Ty}$ be

called *compatible* iff it satisfies the following rules:

$$
\begin{array}{l}
(\sigma\ \mathcal{R}\ \sigma') \ \Rightarrow\ ((\sigma \to \tau)\ \mathcal{R}\ (\sigma' \to \tau)) \\
(\tau\ \mathcal{R}\ \tau') \ \Rightarrow\ ((\sigma \to \tau)\ \mathcal{R}\ (\sigma \to \tau')) \\
((\tau_j\ \mathcal{R}\ \tau_j') \wedge (j \notin I)) \ \Rightarrow\ ((\{i = \tau_i\}^I \cup \{j = \tau_j\})\ \mathcal{R}\ (\{i = \tau_i\}^I \cup \{j = \tau_j'\})) \\
(\tau\ \mathcal{R}\ \sigma) \ \Rightarrow\ ((\forall P.\tau)\ \mathcal{R}\ (\forall P.\sigma))
\end{array}
$$

Definition 7.4 (Type Equivalences)**.** Let $\to_{\mathsf{ty}}$ be the least compatible relation that contains all instances of the rules (1) through (3), below. Let $\to_{\mathsf{ty}}^*$ denote the reflexive and transitive closure of $\to_{\mathsf{ty}}$, and $\leftrightarrow_{\mathsf{ty}}$ the least compatible equivalence relation that contains all instances of (1) through (3).

$$\forall *.\tau \quad \mathcal{R} \quad \tau \tag{1}$$

$$\forall \{i = P_i\}^I.\{i = \tau_i\}^I \quad \mathcal{R} \quad \{i = \forall P_i.\tau_i\}^I \tag{2}$$

$$\{i = \sigma_i\}^I \to \{i = \tau_i\}^I \quad \mathcal{R} \quad \{i = \sigma_i \to \tau_i\}^I \tag{3}$$

The relation $\to_{\mathsf{ty}}$ is a rewriting relation, so the usual terminology of rewriting (redex, contract, contractum, etc.) can be used with $\to_{\mathsf{ty}}$. A type τ is called *irreducible* if there is no type σ such that $\tau \to_{\mathsf{ty}} \sigma$. For any type τ, let $\mathsf{nf}(\tau)$ denote the unique irreducible type σ such that $\tau \to_{\mathsf{ty}}^* \sigma$. (Existence and uniqueness of $\mathsf{nf}(\tau)$ are proved in Lemma 7.9 below.) □

Lemma 7.5.

1. *If* $(\tau \to_{\mathsf{ty}} \sigma)$ *and* $(\tau : \kappa)$*, then* $(\sigma : \kappa)$*.*
2. *If* $(\tau \leftrightarrow_{\mathsf{ty}} \sigma)$*,* $(\tau : \kappa)$ *and* $(\sigma : \kappa')$*, then* $\kappa = \kappa'$*.* □

Lemma 7.6. *If* $(\{i = \tau_i\}^I \to_{\mathsf{ty}}^* \sigma)$*, then* σ *is of the form* $\{i = \sigma_i\}^I$ *where* $\tau_i \to_{\mathsf{ty}}^* \sigma_i$ *for all* $i \in I$*.* □

Lemma 7.7 (Termination)**.** *There is no infinite sequence* $(\tau_n)_{n\in\mathbb{N}}$ *such that* $\tau_n \to_{\mathsf{ty}} \tau_{n+1}$ *for all* $n \in \mathbb{N}$*.* □

Proof. Define a weight function $\|\cdot\|$ on types by

$$\|\tau\| \quad = \quad \left(\begin{array}{ll} & \text{(no. of occurrences of labels in } \tau) \\ + & \text{(no. of occurrences of } * \text{ in } \tau) \end{array}\right)$$

An inspection of the rewriting rules shows that $\tau \to_{\mathsf{ty}} \sigma$ implies $\|\tau\| > \|\sigma\|$. Therefore, every rewriting sequence is finite. □

Lemma 7.8 (Confluence)**.**

1. *If* $\tau_1 \to_{\mathsf{ty}} \tau_2$ *and* $\tau_1 \to_{\mathsf{ty}} \tau_3$ *where* $\tau_2 \neq \tau_3$*, then there is a type* τ_4 *such that* $\tau_2 \to_{\mathsf{ty}} \tau_4$ *and* $\tau_3 \to_{\mathsf{ty}} \tau_4$*.*
2. *If* $\tau_1 \to_{\mathsf{ty}}^* \tau_2$ *and* $\tau_1 \to_{\mathsf{ty}}^* \tau_3$*, then there is a type* τ_4 *such that* $\tau_2 \to_{\mathsf{ty}}^* \tau_4$ *and* $\tau_3 \to_{\mathsf{ty}}^* \tau_4$*.* □

Proof Sketch. An inspection of the rewriting rules shows the following: Whenever a redex τ contains another redex σ, then σ still occurs in the contractum of τ. Moreover, σ does not get duplicated in the contraction step. For this reason, statement (1) holds. Statement (2) follows from (1) by the usual argument. Note that Lemma 7.7 is not needed because of the strength of (1). □

Lemma 7.9 (Unique Normal Forms)**.**
For every type τ *there is a unique irreducible type* σ *such that* $\tau \to_{\mathsf{ty}}^* \sigma$*.* □

Proof. Uniqueness follows from confluence and existence from termination by standard arguments. □

Lemma 7.10. $(\tau \leftrightarrow_{\mathsf{ty}} \sigma)$ *if and only if* $(\mathsf{nf}(\tau) = \mathsf{nf}(\sigma))$*.* □

Proof. This follows immediately from Lemma 7.9. □

7.1.4 A Grammar for Normal Types

An important property of irreducible types is that their top-level structure reflects the top-level structure of their branching shapes. In particular, if a type is irreducible and individual, then it is not of the form $\{i = \tau_i\}^I$. The following grammatical characterization helps make this precise.

Let $\bar{P}^\bullet$ range over $(\mathsf{IndParameter} - \{*\})$ (i.e., over individual type selection parameters of the form $\mathsf{join}\{i = \bar{P}_i\}^I$). The sets $\mathsf{NormalTy}$ of *normal types* and $\mathsf{IndNormalTy}$ of *individual normal types* are defined by the following pseudo-grammars:

$$\begin{array}{rcl} \mu, \nu \in \mathsf{NormalTy} & ::= & \bar{\mu} \mid \{i = \mu_i\}^I \\ \bar{\mu}, \bar{\nu} \in \mathsf{IndNormalTy} & ::= & \alpha \mid \bar{\mu} \to \bar{\nu} \mid \forall \bar{P}^\bullet.\mu \end{array}$$

Let $\mathsf{normal}(\tau)$ abbreviate the statement that τ is normal. By Lemma 7.12 below, normality and irreducibility are equivalent for well formed types.

Lemma 7.11. *If $(\tau : \{i = \kappa_i\}^I)$ and τ is irreducible, then τ is of the form $\{i = \tau_i\}^I$ where $\tau_i : \kappa_i$ for all $i \in I$.* □

Proof. By induction on the derivation of $(\tau : \{i = \kappa_i\}^I)$. □

Lemma 7.12.

1. *If $(\mu \in \mathsf{NormalTy})$, then $(\mu \in \mathsf{Ty})$.*
2. *If $(\bar{\mu} \in \mathsf{IndNormalTy})$ and $\bar{\mu}$ is well formed (i.e., $\bar{\mu} : \kappa$ for some κ), then $(\bar{\mu} \in \mathsf{IndTy})$ (i.e., $\kappa = *$).*
3. *If $(\tau \in \mathsf{IndTy})$ (i.e., $\tau : *$) and $(\tau \in \mathsf{NormalTy})$, then $(\tau \in \mathsf{IndNormalTy})$.*
4. *If μ is normal, then μ is irreducible.*
5. *If τ is well formed and irreducible, then τ is normal.* □

7.1.5 Type Matching

For type checking in λ^{B}, it is necessary to have an algorithm that decides whether a given type σ matches a function type, i.e., whether there are types τ, τ' such that $(\sigma \leftrightarrow_{\mathsf{ty}} \tau \to \tau')$. Lemma 7.14(2) gives a simple decision algorithm for this question. Lemma 7.13(2) says that all possible ways of matching a function type are equivalent. In addition to function types, we also establish similar statements for branching types and $\forall$-types. The results of this section can easily be combined to make a general matching algorithm for $\leftrightarrow_{\mathsf{ty}}$, although we do not do so because we do not need such an algorithm in this paper.

Lemma 7.13.

1. *If $(\{i = \tau_i\}^I \leftrightarrow_{\mathsf{ty}} \{i = \sigma_i\}^I)$, then $(\tau_i \leftrightarrow_{\mathsf{ty}} \sigma_i)$ for all $i \in I$.*
2. *If $(\tau \to \tau' \leftrightarrow_{\mathsf{ty}} \sigma \to \sigma')$, then $(\tau \leftrightarrow_{\mathsf{ty}} \tau')$ and $(\sigma \leftrightarrow_{\mathsf{ty}} \sigma')$.*
3. *If $(\forall P.\tau \leftrightarrow_{\mathsf{ty}} \forall P.\tau')$, then $(\tau \leftrightarrow_{\mathsf{ty}} \tau')$.* □

Proof. Statement (1) follows from Lemmas 7.10 and 7.6. Statement (2) is proved by induction on the size of $(\tau \to \tau')$'s $\to_{\mathsf{ty}}$-rewriting graph. Statement (3) is proved by induction on the size of $(\forall P.\tau)$'s $\to_{\mathsf{ty}}$-rewriting graph. □

Let the partial functions tdom and tcodom from Ty to Ty be defined inductively by the following equations:

$$\begin{array}{rcl} \mathsf{tdom}(\bar{\mu} \to \bar{\nu}) & = & \bar{\mu} \\ \mathsf{tdom}(\{i = \mu_i\}^I) & = & \{i = \mathsf{tdom}(\mu_i)\}^I \\ \mathsf{tdom}(\tau) & = & \mathsf{tdom}(\mathsf{nf}(\tau)) \quad \text{if } \neg\mathsf{normal}(\tau) \end{array}$$

$$\begin{array}{rcl} \mathsf{tcodom}(\bar{\mu} \to \bar{\nu}) & = & \bar{\nu} \\ \mathsf{tcodom}(\{i = \mu_i\}^I) & = & \{i = \mathsf{tcodom}(\mu_i)\}^I \\ \mathsf{tcodom}(\tau) & = & \mathsf{tcodom}(\mathsf{nf}(\tau)) \quad \text{if } \neg\mathsf{normal}(\tau) \end{array}$$

The partial function split from $\mathsf{Parameter} \times \mathsf{Ty}$ to Ty is defined inductively by the equations below. The operation $\mathsf{split}(P, \tau)$ attempts to split the type selection parameter P off the type τ and to return the remaining type.

$$\begin{array}{rcl} \mathsf{split}(*, \mu) & = & \mu \\ \mathsf{split}(\bar{P}^\bullet, \forall \bar{P}^\bullet.\mu) & = & \mu \\ \mathsf{split}(\{i = P_i\}^I, \{i = \mu_i\}^I) & = & \{i = \mathsf{split}(P_i, \mu_i)\}^I \\ \mathsf{split}(P, \tau) & = & \mathsf{split}(P, \mathsf{nf}(\tau)) \quad \text{if } \neg\mathsf{normal}(\tau) \end{array}$$

Lemma 7.14. *Suppose σ is well formed.*

1. *If $(\sigma \leftrightarrow_{\mathsf{ty}} \{i = \tau_i\}^I)$, then $\mathsf{nf}(\sigma)$ is of the form $\{i = \sigma_i\}^I$.*
2. (a) *$\mathsf{tdom}(\sigma)$ is defined if and only if $\mathsf{tcodom}(\sigma)$ is defined.*
   (b) *If $\mathsf{tdom}(\sigma)$ is defined, then $(\sigma \leftrightarrow_{\mathsf{ty}} \mathsf{tdom}(\sigma) \to \mathsf{tcodom}(\sigma))$.*
   (c) *If $(\sigma \leftrightarrow_{\mathsf{ty}} \tau \to \tau')$, then $\mathsf{tdom}(\sigma)$ is defined.*
3. (a) *If $\mathsf{split}(P, \sigma)$ is defined, then $(\sigma \leftrightarrow_{\mathsf{ty}} \forall P. \mathsf{split}(P, \sigma))$.*
   (b) *If $(\sigma \leftrightarrow_{\mathsf{ty}} \forall P.\tau)$ and $\forall P.\tau$ is well-formed, then $\mathsf{split}(P, \sigma)$ is defined.* □

Proof. (1): Follows from Lemmas 7.10 and 7.6.

(2): Part (a) is obvious. Part (b) is easily proved by induction on the definition of tdom. We prove part (c). It suffices to prove (c) for the case where σ, τ, τ' are normal. Let's suppose that this is the case. The proof is by induction on σ's normal form structure:

Case, $(\sigma = \alpha)$**:** The implication holds vacuously, because terms equivalent to α are of the form $(\forall * \ldots. \forall *. \alpha)$.

Case, $(\sigma = \bar{\mu} \to \bar{\nu})$**:** Trivial.

Case, $(\sigma = \forall \bar{P}^\bullet.\mu)$**:** The implication holds vacuously, because terms equivalent to $\forall \bar{P}^\bullet.\mu$ are of the form $(\forall * \ldots. \forall *. \forall \bar{P}^\bullet.\sigma')$.

Case, $(\sigma = \{i = \mu_i\}^I)$**:** Suppose $(\sigma \leftrightarrow_{\mathsf{ty}} \tau \to \tau')$. Then $(\sigma = \mathsf{nf}(\tau \to \tau'))$, by Lemma 7.10. Because τ, τ' are assumed to be in normal form already, it must then be the case that τ, τ' are of the forms $(\tau = \{i = \tau_i\}^I)$ and $(\tau' = \{i = \tau'_i\}^I)$, such that $(\mu_i \leftrightarrow_{\mathsf{ty}} \tau_i \to \tau'_i)$ for all $i \in I$. Then, $\mathsf{tdom}(\mu_i)$ is defined for all $i \in I$, by induction hypothesis. Then, $\mathsf{tdom}(\sigma)$ is defined, by definition of tdom.

(3): Part (a) is easily proved by an induction on the definition of split. We prove part (b). It suffices to prove (b) for the case where σ, τ are normal. Let's suppose that this is the case. The proof is by induction on the structure of P:

Case, $(P = *)$**:** Trivial.

Case, $(P = \bar{P}^\bullet)$ **:** Suppose $(\sigma \leftrightarrow_{\mathsf{ty}} \forall \bar{P}^\bullet.\tau)$. Then σ is of the form $(\forall * \ldots. \forall *. \forall \bar{P}^\bullet.\tau')$. Because σ is normal, $(\sigma = \forall \bar{P}^\bullet.\tau')$. Then $(\mathsf{split}(\bar{P}^\bullet, \sigma) = \tau')$.

Case, $(P = \{i = P_i\}^I)$**:** Suppose σ and $\forall P.\tau$ are well formed and $(\sigma \leftrightarrow_{\mathsf{ty}} \forall P.\tau)$. Because $\forall P.\tau$ is well-formed, we have $(\tau : \lceil P \rceil)$ and $(\forall P.\tau : \lfloor P \rfloor)$. Moreover, τ is normal, by assumption. From τ's normality and $(\tau : \lceil P \rceil)$, it follows that τ is of the form $\{i = \tau_i\}^I$. From $(\forall P.\tau : \lfloor P \rfloor)$ and $(\sigma \leftrightarrow_{\mathsf{ty}} \forall P.\tau)$, it follows that $(\sigma : \lfloor P \rfloor)$, by Lemma 7.5. Then, because σ is normal, it is the case that σ is of the form $\{i = \mu_i\}^I$. Because $(\sigma \leftrightarrow_{\mathsf{ty}} \forall P.\tau \leftrightarrow_{\mathsf{ty}}$

$\{i = \forall P_i.\tau_i\}^I$), it is the case that ($\mu_i \leftrightarrow_{\mathsf{ty}} \forall P_i.\tau_i$) for all $i \in I$, by Lemma 7.13(1). Then, $\mathsf{split}(P_i, \mu_i)$ is defined for all $i \in I$, by induction hypothesis. Then, $\mathsf{split}(P, \sigma)$ is defined, by definition of split. □

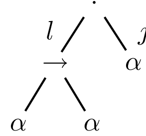
7.2 Expansion and Selection for Types

7.2.1 Type Expansion

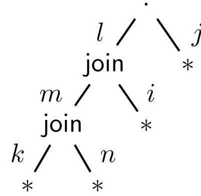
For the typing rules, we need to define some auxiliary operations on types. Let the partial function expand from Ty × Parameter to Ty be inductively defined by the equations below. Applying expand to the pair (τ, P) adjusts the type τ of branching shape $\lfloor P \rfloor$ to the new branching shape $\lceil P \rceil$ (of which $\lfloor P \rfloor$ is a prefix) by duplicating subterms of τ. The duplication is caused by the second of the defining equations.

$$\begin{array}{rcl}
\mathsf{expand}(\mu, *) & = & \mu \\
\mathsf{expand}(\mu, \mathsf{join}\{i = \bar{P}_i\}^I) & = & \{i = \mathsf{expand}(\mu, \bar{P}_i)\}^I \\
\mathsf{expand}(\{i = \mu_i\}^I, \{i = P_i\}^I) & = & \{i = \mathsf{expand}(\mu_i, P_i)\}^I \\
\mathsf{expand}(\tau, P) & = & \mathsf{expand}(\mathsf{nf}(\tau), P) \quad \text{if } \neg\mathsf{normal}(\tau)
\end{array}$$

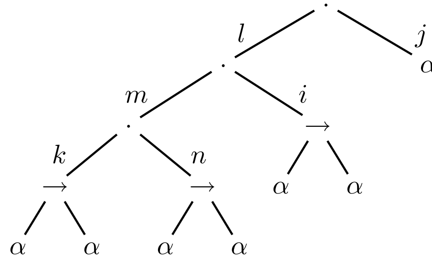
Example 7.15. Suppose $\tau = \{l = \alpha \to \alpha,\ j = \alpha\}$. This type is depicted by the following tree:



Suppose, furthermore, that P is the type selection parameter corresponding to the following tree:



Then $\mathsf{expand}(\tau, P)$ is represented by the following tree:



□

Lemma 7.16. *If* ($\mathsf{expand}(\tau, P) = \tau'$), *then* τ' *is normal.* □

Lemma 7.17.

1. *If* ($\lfloor P \rfloor \leq \kappa$) *and* ($\tau : \kappa$), *then* $\mathsf{expand}(\tau, P)$ *is defined.*
2. ($\tau : \lfloor P \rfloor$) *if and only if* ($\mathsf{expand}(\tau, P) : \lceil P \rceil$). □

Proof. To prove part (1), one first proves the statement for the case where τ is normal, by induction on the structure of P and using Lemma 7.11. Then, for the case where τ is not normal, one uses the fact that normalization preserves kinds (Lemma 7.5). The proof of part (2) follows the same strategy. □

Lemma 7.18. *If* $(\mathsf{expand}(\tau, P) \leftrightarrow_{\mathsf{ty}} \mathsf{expand}(\tau', P))$*, then* $(\tau \leftrightarrow_{\mathsf{ty}} \tau')$*.* □

Proof. It suffices to prove the statement for the case where τ and τ' are normal. For this case, it is proved by induction on the structure of P. □

Lemma 7.19.

1. *If* $\mathsf{expand}(\{i = \tau_i\}^I, P)$ *is defined, then* P *is of the form* $\{i = P'_i\}^I$*.*
2. $\mathsf{expand}(\{i = \tau_i\}^I, \{i = P_i\}^I) \ \leftrightarrow^{\perp}_{\mathsf{ty}} \ \{i = \mathsf{expand}(\tau_i, P_i)\}^I$*.*
3. $\mathsf{expand}(\sigma \to \tau, P) \ \leftrightarrow^{\perp}_{\mathsf{ty}} \ (\mathsf{expand}(\sigma, P) \to \mathsf{expand}(\tau, P))$*.* □

Proof. Parts (1) and (2) are obvious for normal types. For non-normal types, they follow from Lemma 7.6. For normal types, part (3) is proved by induction on the structure of P. For non-normal types, part (3) is proved by induction on the size of the $\to_{\mathsf{ty}}$-rewriting-graph of $(\sigma \to \tau)$, using parts (1) and (2). □

7.2.2 Type Selection Arguments and Selection for Types

The sets $\mathsf{Argument}$ of *type selection arguments* and $\mathsf{IndArgument}$ of *individual type selection arguments* are defined by the following pseudo-grammar:

$$\begin{array}{rcl} A \in \mathsf{Argument} & ::= & \bar{A} \ | \ \{i = A_i\}^I \\ \bar{A} \in \mathsf{IndArgument} & ::= & * \ | \ (i, \bar{A}) \end{array}$$

We define two relations that assign kinds to arguments:

$$\frac{}{\bar{A} : *} \qquad \frac{A_i : \kappa_i \text{ for all } i \in I}{\{i = A_i\}^I : \{i = \kappa_i\}^I}$$

$$\frac{}{* \lhd \kappa} \qquad \frac{\bar{A} \lhd \kappa_j}{j, \bar{A} \lhd \{i = \kappa_i\}^I} \text{ if } j \in I \qquad \frac{A_i \lhd \kappa_i \text{ for all } i \in I}{\{i = A_i\}^I \lhd \{i = \kappa_i\}^I}$$

Note that for every argument A there is exactly one kind κ such that $(A : \kappa)$. On the other hand, there are many kinds κ such that $(A \lhd \kappa)$.

Lemma 7.20. *If* $(A \lhd \kappa)$ *and* $(\kappa \leq \kappa')$*, then* $(A \lhd \kappa')$*.* □

We define two partial functions $\mathsf{select}^{\mathsf{i}}$ and $\mathsf{select}^{\mathsf{b}}$, both from $\mathsf{Ty} \times \mathsf{Argument}$ to Ty. The two functions are similar. The main difference is that $\mathsf{select}^{\mathsf{i}}$ performs selections on individual (joined) types, whereas $\mathsf{select}^{\mathsf{b}}$ performs selections on branching types. Another difference is that $\mathsf{select}^{\mathsf{b}}(\mu, *)$ is always defined, whereas $\mathsf{select}^{\mathsf{i}}(\mu, *)$ is only defined if μ is an individual type.

$$\begin{array}{rcl} \mathsf{select}^{\mathsf{i}}(\bar{\mu}, *) & = & \bar{\mu} \\ \mathsf{select}^{\mathsf{i}}(\forall(\mathsf{join}\{i = \bar{P}_i\}^I).\{i = \mu_i\}^I, (j, \bar{A})) & = & \mathsf{select}^{\mathsf{i}}(\forall \bar{P}_j.\mu_j, \bar{A}), \quad \text{if } (j \in I) \\ \mathsf{select}^{\mathsf{i}}(\{i = \mu_i\}^I, \{i = A_i\}^I) & = & \{i = \mathsf{select}^{\mathsf{i}}(\mu_i, A_i)\}^I \\ \mathsf{select}^{\mathsf{i}}(\tau, A) & = & \mathsf{select}^{\mathsf{i}}(\mathsf{nf}(\tau), A) \quad \text{if } \neg\mathsf{normal}(\tau) \end{array}$$

$$\begin{array}{rcl} \mathsf{select}^{\mathsf{b}}(\mu, *) & = & \mu \\ \mathsf{select}^{\mathsf{b}}(\{i = \mu_i\}^I, (j, \bar{A})) & = & \mathsf{select}^{\mathsf{b}}(\mu_j, \bar{A}), \quad \text{if } (j \in I) \\ \mathsf{select}^{\mathsf{b}}(\{i = \mu_i\}^I, \{i = A_i\}^I) & = & \{i = \mathsf{select}^{\mathsf{b}}(\mu_i, A_i)\}^I \\ \mathsf{select}^{\mathsf{b}}(\tau, A) & = & \mathsf{select}^{\mathsf{b}}(\mathsf{nf}(\tau), A) \quad \text{if } \neg\mathsf{normal}(\tau) \end{array}$$

Lemma 7.21. *If* $f \in \{\mathsf{select}^{\mathsf{i}}, \mathsf{select}^{\mathsf{b}}\}$ *and* $f(\tau, A) = \tau'$, *then* τ' *is normal.* □

Lemma 7.22.

1. *If* $(\tau : \kappa)$ *and* $\mathsf{select}^{\mathsf{i}}(\tau, A)$ *is defined, then* $(A : \kappa)$.
2. *If* $(\tau : \kappa)$ *and* $(\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$, *then* $(\tau' : \kappa)$.
3. *If* $(\tau : \kappa)$, *then* $(A \triangleleft \kappa)$ *if and only if* $\mathsf{select}^{\mathsf{b}}(\tau, A)$ *is defined.* □

Proof. Each statement, separately, by induction on the structure of A. □

Lemma 7.23. *Let* f *range over* $\{\mathsf{select}^{\mathsf{i}}, \mathsf{select}^{\mathsf{b}}\}$.

1. *If* $\mathsf{select}^{\mathsf{i}}(\{i = \tau_i\}^I, A)$ *is defined, then* A *is of the form* $\{i = A'_i\}^I$.
2. *If* $\mathsf{select}^{\mathsf{b}}(\{i = \tau_i\}^I, A)$ *is defined, then one of the following statements holds:*
   (a) A *is of the form* $(A = \{i = A'_i\}^I)$.
   (b) *There is a label* j *in* I *and an individual type selection argument* $\bar{A}$ *such that* $A = (j, \bar{A})$.
3. $f(\{i = \tau_i\}^I, \{i = A_i\}^I) \leftrightarrow^{\perp}_{\mathsf{ty}} \{i = f(\tau_i, A_i)\}^I$.
4. *If* $(j \in I)$ *and* $(P = \mathsf{join}\{i = \bar{P}_i\}^I)$, *then* $(\mathsf{select}^{\mathsf{i}}(P, (j, \bar{A})) \leftrightarrow^{\perp}_{\mathsf{ty}} \mathsf{select}^{\mathsf{i}}(\forall \bar{P}_j.\tau_j, \bar{A}))$.
5. *If* $(j \in I)$, *then* $(\mathsf{select}^{\mathsf{b}}(\{i = \tau_i\}^I, (j, \bar{A})) \leftrightarrow^{\perp}_{\mathsf{ty}} \mathsf{select}^{\mathsf{b}}(\tau_j, \bar{A}))$.
6. $(f(\sigma \to \tau, A) \leftrightarrow^{\perp}_{\mathsf{ty}} f(\sigma, A) \to f(\tau, A))$. □

Proof. The proof follows the same strategy as the proof of Lemma 7.19. □

7.3 Trivial Parameters and Arguments

A type selection parameter or argument is called *trivial* if it is also a kind.

Lemma 7.24 (Trivial Parameters and Arguments)**.**

1. *If* P *is a trivial parameter and* $(\tau : P)$, *then* $(\forall P.\tau \leftrightarrow_{\mathsf{ty}} \tau)$.
2. *If* P *is a trivial parameter and* $(\tau : P)$, *then* $(\mathsf{expand}(\tau, P) \leftrightarrow_{\mathsf{ty}} \tau)$.
3. *If* P *is a trivial parameter, then* $(\lfloor P \rfloor = \lceil P \rceil = P)$.
4. *If* A *is a trivial argument and* $(\tau : A)$, *then* $(\mathsf{select}^{\mathsf{i}}(\tau, A) \leftrightarrow_{\mathsf{ty}} \tau)$.
5. *If* A *is a trivial argument and* $(\tau : \kappa)$ *and* $(A \triangleleft \kappa)$, *then* $(\mathsf{select}^{\mathsf{b}}(\tau, A) \leftrightarrow_{\mathsf{ty}} \tau)$. □

Proof. The first three statements are proved by induction on the structure of P, the last two by induction on the structure of A. □

7.4 Terms and Typing Rules

The set Term of λ^{B}-terms is defined by the following pseudo-grammar:

$$M, N \in \mathsf{Term} \quad ::= \quad \Lambda P.M \;\mid\; M[A] \;\mid\; \lambda x^{\tau}.M \;\mid\; M\ N \;\mid\; x^{\tau}$$

The λx binds the variable x in the usual way. We use the usual notion of free variables of a term. We use the usual notion of α-conversion for renaming of bound variables and identify terms that are α-equivalent. We use the following parsing conventions: $M\,N$ binds more tightly than $M[A]$ binds more tightly than both $\Lambda P.M$ and $\lambda x^{\tau}.M$; and $M\,N$ associates to the left. A *type environment* is defined to be a finite function from Var to Ty. Let the metavariable E range over type environments. Let the definitions of kind assignment, type equivalence, and expansion be extended to type environments as follows:

$$E : \kappa \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad E(x) : \kappa \text{ for all } x \in \mathsf{dom}(E)$$

$$E \leftrightarrow_{\mathsf{ty}} E' \quad \stackrel{\text{def}}{\Longleftrightarrow} \quad \begin{cases} \mathsf{dom}(E) = \mathsf{dom}(E') \text{ and} \\ (E(x) \leftrightarrow_{\mathsf{ty}} E'(x)) \text{ for all } x \in \mathsf{dom}(E) \end{cases}$$

Let $\mathsf{expand}(E, P)(x)$ be defined iff $\mathsf{expand}(E(x), P)$ is defined for all $x \in \mathsf{dom}(E)$. In this case, it is defined by

$$\mathsf{expand}(E, P)(x) \quad \stackrel{\text{def}}{=\!=} \quad \mathsf{expand}(E(x), P).$$

Typing judgments are of the form:

$$E \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa$$

The valid typing judgments are those that can be proven using the typing rules in Figure 5.

$$(\mathsf{ax}) \qquad \frac{}{E \vdash^{\mathsf{B}} x^{\tau} : \tau \ \mathsf{at}\ \kappa} \quad \text{if } (E : \kappa) \text{ and } (\tau \leftrightarrow_{\mathsf{ty}} E(x))$$

$$(\rightarrow_{\mathsf{i}}) \qquad \frac{E[x \mapsto \sigma] \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa}{E \vdash^{\mathsf{B}} \lambda x^{\sigma}.M : \sigma \rightarrow \tau \ \mathsf{at}\ \kappa}$$

$$(\rightarrow_{\mathsf{e}}) \qquad \frac{E \vdash^{\mathsf{B}} M : \sigma \rightarrow \tau \ \mathsf{at}\ \kappa; \quad E \vdash^{\mathsf{B}} N : \sigma \ \mathsf{at}\ \kappa}{E \vdash^{\mathsf{B}} M\ N : \tau \ \mathsf{at}\ \kappa}$$

$$(\forall_{\mathsf{i}}) \qquad \frac{\mathsf{expand}(E, P) \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \lceil P \rceil}{E \vdash^{\mathsf{B}} \Lambda P.M : \forall P.\tau \ \mathsf{at}\ \lfloor P \rfloor}$$

$$(\forall_{\mathsf{e}}) \qquad \frac{E \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa}{E \vdash^{\mathsf{B}} M[A] : \tau' \ \mathsf{at}\ \kappa} \quad \text{if } (\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$$

$$(\leftrightarrow_{\mathsf{ty}}) \qquad \frac{E \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa}{E \vdash^{\mathsf{B}} M : \tau' \ \mathsf{at}\ \kappa} \quad \text{if } (\tau \leftrightarrow_{\mathsf{ty}} \tau') \text{ and } (\tau' : \kappa)$$

Figure 5: λ^{B} — typing rules

Lemma 7.25.

1. *If* $(E \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa)$*, then* $(E : \kappa)$ *and* $(\tau : \kappa)$*.*
2. *If* $(E \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa)$ *and* $(E \leftrightarrow_{\mathsf{ty}} E')$*, then* $(E' \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa)$*.* □

Proof. Statement (1) is proved by induction on the derivation of ($E \vdash^{\mathsf{B}} M : \tau$ at κ), using the fact that $\mathsf{select}^{\mathsf{i}}$ preserves kinds (Lemma 7.22), and that expand reflects kinds in the way expressed in Lemma 7.17. Statement (2) is proved by induction on the structure of ($E \vdash^{\mathsf{B}} M : \tau$ at κ). ☐

Proposition 7.26 (Unicity of Typing). *If* ($E \vdash^{\mathsf{B}} M : \tau$ at κ) *and* ($E' \vdash^{\mathsf{B}} M : \tau'$ at κ'), *then* $(\tau \leftrightarrow_{\mathsf{ty}} \tau')$, $(\kappa = \kappa')$ *and* $(E(x) \leftrightarrow_{\mathsf{ty}} E'(x))$ *for all free variables* x *of* M. ☐

Proof. By induction on the structure of M, using the fact that expand reflects type equivalence (Lemma 7.18). ☐

Example 7.27. Consider

$$
\begin{array}{ll}
P \;=\; \mathsf{join}\{i = *,\, h = *\}, & \sigma \;=\; \forall P.\{i = \alpha \to \alpha,\, h = \beta \to \beta\}, \\
P' \;=\; \mathsf{join}\{j = *,\, l = *\}, & \sigma'' \;=\; \forall P'.\{j = \alpha \to \alpha,\, l = \beta \to \beta\} \\
\sigma' \;=\; \{j = \sigma,\, l = \sigma\}, & \tau' \;=\; \{j = \alpha,\, l = \beta\}, \\
A \;=\; \{j = (i, *),\, l = (h, *)\}, & M = \lambda y^{\sigma}.\Lambda P'.\lambda x^{\tau'}.\, y^{\sigma'}\ [A]\ x^{\tau'}.
\end{array}
$$

The type erasure of M (defined in Section 9.1) is $(\lambda y.\lambda x.\, y\, x)$. This untyped term can be given the type $(\wedge\{i = \alpha \to \alpha,\, h = \beta \to \beta\} \to \wedge\{j = \alpha \to \alpha,\, l = \beta \to \beta\})$ in λ^{I}. Correspondingly, the judgment

$$\vdash^{\mathsf{B}}\ M\ :\ (\sigma \to \sigma'')\ \ \mathsf{at}\ \ *$$

is derivable in our system λ^{B}. What follows is the derivation of this judgment, presented in a goal-directed style. Note that in going from goal (2) to goal (3), the type σ of y is expanded to σ'. In the presentation of the derivation, we have omitted side conditions. Most notably, going from goal (6.l) to goal (6.l.1) is valid because $(\mathsf{select}^{\mathsf{i}}(\sigma', A) = \tau' \to \tau')$. Moreover, the axioms (6.r) and (6.l.1) hold because they satisfy the side condition for (ax).

1.	$\vdash^{\mathsf{B}}\ M\ :\ (\sigma \to \sigma'')$ at $*$	By $(\to_{\mathsf{i}})$
2.	$y : \sigma\ \vdash^{\mathsf{B}}\ (\Lambda P'.\lambda x^{\tau'}.\, y^{\sigma'}\ [A]\ x^{\tau'})\ :\ \sigma''$ at $*$	By $(\forall_{\mathsf{i}})$
3.	$y : \sigma'\ \vdash^{\mathsf{B}}\ (\lambda x^{\tau'}.\, y^{\sigma'}\ [A]\ x^{\tau'})\ :$ $\{j = \alpha \to \alpha,\, l = \beta \to \beta\}$ at $\{j = *,\, l = *\}$	By $(\leftrightarrow_{\mathsf{ty}})$
4.	$y : \sigma'\ \vdash^{\mathsf{B}}\ (\lambda x^{\tau'}.\, y^{\sigma'}\ [A]\ x^{\tau'})\ :\ (\tau' \to \tau')$ at $\{j = *,\, l = *\}$	By $(\to_{\mathsf{i}})$
5.	$y : \sigma',\, x : \tau'\ \vdash^{\mathsf{B}}\ (y^{\sigma'}\ [A]\ x^{\tau'})\ :\ \tau'$ at $\{j = *,\, l = *\}$	By $(\to_{\mathsf{e}})$
6.r.	$y : \sigma',\, x : \tau'\ \vdash^{\mathsf{B}}\ x^{\tau'}\ :\ \tau'$ at $\{j = *,\, l = *\}$	By (ax)
6.l.	$y : \sigma',\, x : \tau'\ \vdash^{\mathsf{B}}\ (y^{\sigma'}\ [A])\ :\ (\tau' \to \tau')$ at $\{j = *,\, l = *\}$	By $(\forall_{\mathsf{e}})$
6.l.1	$y : \sigma',\, x : \tau'\ \vdash^{\mathsf{B}}\ y^{\sigma'}\ :\ \sigma'$ at $\{j = *,\ l = *\}$	By (ax)

☐

8 Expansion and Selection for Terms

In this section, we extend the expansion and selection operations to terms. Term expansion and selection are of technical importance. Term expansion will be used in our definition of term substitution. Selection for terms will be used in the definition of a term rewriting rule, and also when we show that type erasure maps well-typed λ^{B}-terms to well-typed λ^{I}-terms.

In Lemmas 8.2 and 8.5, we collect useful properties of term expansion and selection. Importantly, both expand and $\mathsf{select}^{\mathsf{b}}$ preserve typing judgments, as stated in Lemmas 8.4 and 8.6.

8.1 Term Expansion

In order to later define substitution and β-reduction in a way that preserves types, we need to extend the expand operation to terms. This is necessary because types of free variables get expanded in the typing rule $(\forall_{\mathsf{i}})$. Terms substituted for free variables must be expanded similarly.

First, the partial function expand is extended to parameters, arguments and kinds, inductively by the following equations, where X ranges over $\mathsf{Parameter} \cup \mathsf{Argument} \cup \mathsf{BShape}$:

$$\begin{array}{rcl} \mathsf{expand}(X, *) & = & X \\ \mathsf{expand}(X, \mathsf{join}\{i = \bar{P}_i\}^I) & = & \{i = \mathsf{expand}(X, \bar{P}_i)\}^I \\ \mathsf{expand}(\{i = X_i\}^I, \{i = P_i\}^I) & = & \{i = \mathsf{expand}(X_i, P_i)\}^I \end{array}$$

Now, the partial function expand is inductively extended to terms:

$$\begin{array}{rcl} \mathsf{expand}(\Lambda P'.M, P) & = & \Lambda(\mathsf{expand}(P', P)).\ \mathsf{expand}(M, P) \\ \mathsf{expand}(M[A], P) & = & (\mathsf{expand}(M, P))[\mathsf{expand}(A, P)] \\ \mathsf{expand}(\lambda x^{\tau}.M, P) & = & \lambda x^{\mathsf{expand}(\tau, P)}.\ \mathsf{expand}(M, P) \\ \mathsf{expand}(M\ N, P) & = & (\mathsf{expand}(M, P))\ (\mathsf{expand}(N, P)) \\ \mathsf{expand}(x^{\tau}, P) & = & x^{\mathsf{expand}(\tau, P)} \end{array}$$

Example 8.1. Consider the term $N = \Lambda P.\lambda x^{\tau}.x^{\tau}$, where $P = \mathsf{join}\{i = *,\ h = *\}$ and $\tau = \{i = \alpha,\ h = \beta\}$. Its type erasure (defined in Section 9.1) is $\lambda x.x$. Its type is

$$\sigma \quad = \quad \forall P.\{i = \alpha \to \alpha,\ h = \beta \to \beta\}.$$

Expanding N by the parameter $P' = \mathsf{join}\{j = *, l = *\}$ results in this:

$$\mathsf{expand}(N, P') \ = \ \Lambda\,\{j = P,\ l = P\}\,.\lambda x^{\{j=\tau,\ l=\tau\}}.x^{\{j=\tau,\ l=\tau\}}$$ □

Lemma 8.2 (Properties of expand)**.**

1. *If* $(\lfloor P \rfloor \le \kappa)$*, then* $\mathsf{expand}(\kappa, P)$ *is defined.*
2. *If* $(\lfloor P \rfloor \le \kappa)$ *and* $(\tau : \kappa)$*, then* $(\mathsf{expand}(\tau, P) : \mathsf{expand}(\kappa, P))$.
3. *If* $(\lfloor P \rfloor \le \kappa)$ *and* $(A : \kappa)$*, then* $(\mathsf{expand}(A, P) : \mathsf{expand}(\kappa, P))$.
4. *If* $(\lfloor P \rfloor \le \lfloor P' \rfloor)$*, then* $\mathsf{expand}(P', P)$ *is defined.*
5. *If* $(\lfloor P \rfloor \le \lfloor P' \rfloor)$*, then* $(\mathsf{expand}(\lfloor P' \rfloor, P) = \lfloor \mathsf{expand}(P', P) \rfloor)$.
6. *If* $(\lfloor P \rfloor \le \lfloor P' \rfloor)$*, then* $(\mathsf{expand}(\lceil P' \rceil, P) = \lceil \mathsf{expand}(P', P) \rceil)$.
7. $\lceil P \rceil = \mathsf{expand}(\lfloor P \rfloor, P)$.
8. *If* $(\lfloor P \rfloor \le \lfloor P' \rfloor)$ *and* $(\tau : \lfloor P' \rfloor)$*,*
   then $(\mathsf{expand}(\mathsf{expand}(\tau, P'), P) \leftrightarrow_{\mathsf{ty}} \mathsf{expand}(\mathsf{expand}(\tau, P), \mathsf{expand}(P', P)))$.
9. *If* $(\lfloor P \rfloor \le \lfloor P' \rfloor)$ *and* $(\tau : \lceil P' \rceil)$*,*
   then $(\mathsf{expand}(\forall P'.\tau, P) \leftrightarrow_{\mathsf{ty}} \forall(\mathsf{expand}(P', P)).(\mathsf{expand}(\tau, P)))$.
10. *If* $(\lfloor P \rfloor \le \kappa)$*,* $(\tau : \kappa)$ *and* $\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau'$*,*
    then $(\mathsf{expand}(\tau', P) \ \leftrightarrow_{\mathsf{ty}} \ \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)))$. □

Proof. (1)–(7): Each one separately, by induction on the structure of P.

(8): It suffices to prove the following statement — the general statement easily follows.

If $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, $(\tau : \lfloor P' \rfloor)$ *and* τ *is normal,*
then $(\mathsf{expand}(\mathsf{expand}(\tau, P'), P) \leftrightarrow_{\mathsf{ty}} \mathsf{expand}(\mathsf{expand}(\tau, P), \mathsf{expand}(P', P)))$.

We prove this statement by induction on the structure of P:

Case, $(P = *)$**:** Suppose $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, $(\tau : \lfloor P' \rfloor)$ and τ is normal. By definition of expand,

$$\mathsf{expand}(\mathsf{expand}(\tau, P'), *) \;\leftrightarrow^{\perp}_{\mathsf{ty}}\; \mathsf{expand}(\tau, P') \;\leftrightarrow^{\perp}_{\mathsf{ty}}\; \mathsf{expand}(\mathsf{expand}(\tau, *), \mathsf{expand}(P', *))$$

Both equations hold by definition of expand. Moreover, by statement (2), $\mathsf{expand}(\tau, P')$ is defined.

Case, $(P = \mathsf{join}\{i = \bar{P}_i\})$**:** Suppose $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, $(\tau : \lfloor P' \rfloor)$ and τ is normal.

$$\begin{array}{ll} & \mathsf{expand}(\mathsf{expand}(\tau, P'), P) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \{i = \mathsf{expand}(\mathsf{expand}(\tau, P'), \bar{P}_i)\}^I \\ \leftrightarrow_{\mathsf{ty}} & \{i = \mathsf{expand}(\mathsf{expand}(\tau, \bar{P}_i), \mathsf{expand}(P', \bar{P}_i))\}^I \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{expand}(\{i = \mathsf{expand}(\tau, \bar{P}_i)\}^I, \{i = \mathsf{expand}(P', \bar{P}_i)\}^I) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{expand}(\mathsf{expand}(\tau, P), \mathsf{expand}(P', P)) \end{array}$$

The first, third and fourth of these equations follow from the definition of expand. The second one holds by induction hypotheses, because $(\lfloor \bar{P}_i \rfloor = * \leq \lfloor P' \rfloor)$ for all $i \in I$.

Case, $(P = \{i = P_i\}^I)$**:** Suppose $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, $(\tau : \lfloor P' \rfloor)$ and τ is normal. Because $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, P' is of the form $\{i = P'_i\}^I$ where $\lfloor P_i \rfloor \leq \lfloor P'_i \rfloor$ for all $i \in I$. Because $(\tau : \lfloor P' \rfloor)$ and τ is normal, τ is of the form $\{i = \tau_i\}^I$ where $\tau_i : \lfloor P'_i \rfloor$ for all $i \in I$.

$$\begin{array}{ll} & \mathsf{expand}(\mathsf{expand}(\tau, P'), P) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{expand}(\{i = \mathsf{expand}(\tau_i, P'_i)\}^I, P) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \{i = \mathsf{expand}(\mathsf{expand}(\tau_i, P'_i), P_i)\}^I \\ \leftrightarrow_{\mathsf{ty}} & \{i = \mathsf{expand}(\mathsf{expand}(\tau_i, P_i), \mathsf{expand}(P'_i, P_i))\}^I \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{expand}(\{i = \mathsf{expand}(\tau_i, P_i)\}^I, \{i = \mathsf{expand}(P'_i, P_i)\}^I) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{expand}(\mathsf{expand}(\tau, P), \mathsf{expand}(P', P)) \end{array}$$

The first, second, fourth and fifth of these equations hold by definition of expand, and the third one holds by induction hypotheses.

(9): It suffices to prove the following statement — the general statement easily follows.

If $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, $(\tau : \lceil P' \rceil)$ *and* τ *is normal,*
then $(\mathsf{expand}(\forall P'.\tau, P) \leftrightarrow_{\mathsf{ty}} \forall(\mathsf{expand}(P', P)).(\mathsf{expand}(\tau, P)))$.

We prove the statement by induction on the structure of P:

Case, $(P = *)$**:** Suppose $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, $(\tau : \lceil P' \rceil)$ and τ is normal.

$$\mathsf{expand}(\forall P'.\tau, *) \;\leftrightarrow_{\mathsf{ty}}\; \forall P'.\tau \;\leftrightarrow_{\mathsf{ty}}\; \forall(\mathsf{expand}(P', *)).(\mathsf{expand}(\tau, *))$$

Case, $(P' = *)$**:** Suppose $(\lfloor P \rfloor \leq *)$, $(\tau : *)$ and τ is normal.

$$\mathsf{expand}(\forall *.\tau, P) \;\leftrightarrow^{\perp}_{\mathsf{ty}}\; \mathsf{expand}(\tau, P) \;\leftrightarrow^{\perp}_{\mathsf{ty}}\; \forall(\mathsf{expand}(*, P)).(\mathsf{expand}(\tau, P))$$

The first equation holds by definition of expand, and the second one because $\mathsf{expand}(*, P)$ is a trivial parameter. $\mathsf{expand}(\tau, P)$ is defined, because P is individual.

Case, $(P = \mathsf{join}\{i = \bar{P}_i\})$ and $(P' \neq *)$**:** Suppose $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, $(\tau : \lceil P' \rceil)$ and τ is normal. Then $(\forall P'.\tau)$ is normal.

$$\begin{array}{ll} & \mathsf{expand}(\forall P'.\tau, P) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \{i = \mathsf{expand}(\forall P'.\tau, \bar{P}_i)\}^I \\ \leftrightarrow_{\mathsf{ty}} & \{i = \forall(\mathsf{expand}(P', \bar{P}_i)).(\mathsf{expand}(\tau, \bar{P}_i))\}^I \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \forall\{i = \mathsf{expand}(P', \bar{P}_i)\}^I.\{i = \mathsf{expand}(\tau, \bar{P}_i)\}^I \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \forall(\mathsf{expand}(P', P)).(\mathsf{expand}(\tau, P)) \end{array}$$

The first and fourth of these equations hold by definition of expand. The second one holds by induction hypotheses, because $(\lfloor \bar{P}_i \rfloor = * \leq \lfloor P' \rfloor)$ for all $i \in I$. The third one holds by definition of $\leftrightarrow_{\mathsf{ty}}$.

Case, $(P = \{i = P_i\}^I)$**:** Suppose $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, $(\tau : \lceil P' \rceil)$ and τ is normal. Because $(\lfloor P \rfloor \leq \lfloor P' \rfloor)$, P' is of the form $\{i = P'_i\}^I$ such that $\lfloor P_i \rfloor \leq \lfloor P'_i \rfloor$ for all $i \in I$. Because $(\tau : \lceil P' \rceil)$ and τ is normal, τ is of the form $\{i = \tau_i\}^I$ where $\tau_i : \lceil P'_i \rceil$ for all $i \in I$.

$$\begin{array}{ll} & \mathsf{expand}(\forall P'.\tau, P) \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \mathsf{expand}(\{i = \forall P'_i.\tau_i\}^I, P) \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \{i = \mathsf{expand}(\forall P'_i.\tau_i, P_i)\}^I \\ \leftrightarrow_{\mathsf{ty}} & \{i = \forall(\mathsf{expand}(P'_i, P_i)).(\mathsf{expand}(\tau_i, P_i))\}^I \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \forall\{i = \mathsf{expand}(P', P_i)\}^I.\{i = \mathsf{expand}(\tau_i, P_i)\}^I \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \forall(\mathsf{expand}(P', P)).(\mathsf{expand}(\tau, P)) \end{array}$$

The first and last of these equations hold by definition of expand, the second one by Lemma 7.19, the third one by induction hypotheses and the fourth one by definition of $\leftrightarrow_{\mathsf{ty}}$.

(10): It suffices to prove the following statement — the general statement easily follows.

If $(\lfloor P \rfloor \leq \kappa)$, $(\tau : \kappa)$, $(\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$ *and* τ *is normal,*
then $(\mathsf{expand}(\tau', P) \ \leftrightarrow_{\mathsf{ty}} \ \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)))$.

We prove this statement by induction on the structure of P:

Case, $(P = *)$**:** Suppose $(\lfloor P \rfloor \leq \kappa)$, $(\tau : \kappa)$, $(\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$ and τ is normal.

$$\mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, *), \mathsf{expand}(A, *)) \ \leftrightarrow^{\bot}_{\mathsf{ty}} \ \mathsf{select}^{\mathsf{i}}(\tau, A) \ = \ \tau' \ \leftrightarrow_{\mathsf{ty}} \ \mathsf{expand}(\tau', *)$$

The first and the last of these equations follow from the definition of expand.

Case, $(P = \mathsf{join}\{i = \bar{P}_i\})$**:** Suppose $(\lfloor P \rfloor \leq \kappa)$, $(\tau : \kappa)$, $(\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$ and τ is normal.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)) \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{i}}(\{i = \mathsf{expand}(\tau, \bar{P}_i)\}^I, \{i = \mathsf{expand}(A, \bar{P}_i)\}^I) \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \{i = \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, \bar{P}_i), \mathsf{expand}(A, \bar{P}_i))\}^I \\ \leftrightarrow_{\mathsf{ty}} & \{i = \mathsf{expand}(\tau', \bar{P}_i)\}^I \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \mathsf{expand}(\tau', P) \end{array}$$

The first of these equations follows from the definition of expand, the second one from the definition of $\mathsf{select}^{\mathsf{i}}$, and the last one follows from the definition of expand. The third one holds by induction hypotheses, because $(\lfloor \bar{P}_i \rfloor = * \leq \kappa)$ for all $i \in I$.

Case, $(P = \{i = P_i\}^I)$**:** Suppose $(\lfloor P \rfloor \leq \kappa)$, $(\tau : \kappa)$, $(\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$ and τ is normal. Because $(\lfloor P \rfloor \leq \kappa)$, κ is of the form $\{i = \kappa_i\}^I$ such that $(\lfloor P_i \rfloor \leq \kappa_i)$ for all $i \in I$. Because $(\tau : \kappa)$ and τ is normal, τ is of the form $\{i = \tau_i\}^I$ where $\tau_i : \kappa_i$ for all $i \in I$. Because $(\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$, A and τ' are of the forms $(\tau' = \{i = \tau'_i\}^I)$ and $(A = \{i = A_i\}^I)$ such that $(\mathsf{select}^{\mathsf{i}}(\tau_i, A_i) = \tau'_i)$ for all $i \in I$.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)) \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{i}}(\{i = \mathsf{expand}(\tau_i, P_i)\}^I, \{i = \mathsf{expand}(A_i, P_i)\}^I) \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \{i = \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau_i, P_i), \mathsf{expand}(A_i, P_i))\}^I \\ \leftrightarrow_{\mathsf{ty}} & \{i = \mathsf{expand}(\tau'_i, P_i)\}^I \\ \leftrightarrow^{\bot}_{\mathsf{ty}} & \mathsf{expand}(\tau', P) \end{array}$$

The first of these equations follows from the definition of expand, the second one from the definition of $\mathsf{select}^{\mathsf{i}}$, the third one holds by induction hypotheses, and the last one follows from the definition of expand. □

Lemma 8.3. *If* $(E \vdash^{\mathsf{B}} M : \tau \ \mathtt{at} \ \kappa)$ *and* $(\lfloor P \rfloor \leq \kappa)$,
then $(\mathsf{expand}(E, P) \vdash^{\mathsf{B}} \mathsf{expand}(M, P) : \mathsf{expand}(\tau, P) \ \mathtt{at} \ \mathsf{expand}(\kappa, P))$. □

Proof. By induction on the derivation of $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa)$ The interesting cases are the ones for ΛP-abstraction and $[A]$-application:

Case:

$$\frac{\mathsf{expand}(E, P') \vdash^{\mathsf{B}} M : \tau \text{ at } \lceil P' \rceil}{E \vdash^{\mathsf{B}} \Lambda P'.M : \forall P'.\tau \text{ at } \lfloor P' \rfloor}$$

$\lfloor P \rfloor \leq \lfloor P' \rfloor$ — Assumption

$\lfloor P \rfloor \leq \lceil P' \rceil$ — By $\lfloor P' \rfloor \leq \lceil P' \rceil$ and transitivity of $\leq$

$\mathsf{expand}(\mathsf{expand}(E, P'), P)$
$\quad \vdash^{\mathsf{B}} \mathsf{expand}(M, P) : \mathsf{expand}(\tau, P) \text{ at } \mathsf{expand}(\lceil P' \rceil, P)$ — By ind. hyp.

$\mathsf{expand}(\mathsf{expand}(E, P), \mathsf{expand}(P', P))$
$\quad \vdash^{\mathsf{B}} \mathsf{expand}(M, P) : \mathsf{expand}(\tau, P) \text{ at } \lceil \mathsf{expand}(P', P) \rceil$
By Lemma 8.2, (8) and (6)

$\mathsf{expand}(E, P)$
$\quad \vdash^{\mathsf{B}} \mathsf{expand}(\Lambda P'.M, P) : \forall(\mathsf{expand}(P', P)).(\mathsf{expand}(\tau, P)) \text{ at } \lfloor \mathsf{expand}(P', P) \rfloor$
By $(\forall_{\mathsf{i}})$ and definition of expand for terms

$\mathsf{expand}(E, P)$
$\quad \vdash^{\mathsf{B}} \mathsf{expand}(\Lambda P'.M, P) : \mathsf{expand}(\forall P'.\tau, P) \text{ at } \mathsf{expand}(\lfloor P' \rfloor, P)$
By Lemma 8.2, (9) and (5)

Case:

$$\frac{E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa}{E \vdash^{\mathsf{B}} M[A] : \tau' \text{ at } \kappa} \quad \text{if } (\mathsf{select}^{\mathsf{i}}(\tau, A) = \tau')$$

$\lfloor P \rfloor \leq \kappa$ — Assumption

$\mathsf{expand}(E, P) \vdash^{\mathsf{B}} \mathsf{expand}(M, P) : \mathsf{expand}(\tau, P) \text{ at } \mathsf{expand}(\kappa, P)$ — By ind. hyp.

$\mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)) \leftrightarrow_{\mathsf{ty}} \mathsf{expand}(\tau', P)$ — By Lemma 8.2 (10)

$\mathsf{expand}(E, P) \vdash^{\mathsf{B}} \mathsf{expand}(M[A], P) : \mathsf{expand}(\tau', P) \text{ at } \mathsf{expand}(\kappa, P)$
By $(\forall_{\mathsf{e}})$ and definition of expand for terms

□

Corollary 8.4. *If* $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \lfloor P \rfloor)$,
then $(\mathsf{expand}(E, P) \vdash^{\mathsf{B}} \mathsf{expand}(M, P) : \mathsf{expand}(\tau, P) \text{ at } \lceil P \rceil)$. □

Proof. Follows from Lemma 8.3 and Lemma 8.2 (7). □

8.2 Selection for Terms

We extend $\mathsf{select}^{\mathsf{b}}$ to parameters, arguments and kinds, inductively by the following equations, where X ranges over $\mathsf{Parameter} \cup \mathsf{Argument} \cup \mathsf{BShape}$:

$$\begin{aligned}
\mathsf{select}^{\mathsf{b}}(X, *) &= X \\
\mathsf{select}^{\mathsf{b}}(\{i = X_i\}^I, (j, \bar{A})) &= \mathsf{select}^{\mathsf{b}}(X_j, \bar{A}), \quad \text{if } j \in I \\
\mathsf{select}^{\mathsf{b}}(\{i = X_i\}^I, \{i = A_i\}^I) &= \{i = \mathsf{select}^{\mathsf{b}}(X_i, A_i)\}^I
\end{aligned}$$

The partial function $\mathsf{select}^{\mathsf{b}}$ is inductively extended to terms:

$$\begin{aligned}
\mathsf{select}^{\mathsf{b}}(\Lambda P'.M, A) &= \Lambda(\mathsf{select}^{\mathsf{b}}(P', A)).\ \mathsf{select}^{\mathsf{b}}(M, A) \\
\mathsf{select}^{\mathsf{b}}(M[A'], A) &= (\mathsf{select}^{\mathsf{b}}(M, A))[\mathsf{select}^{\mathsf{b}}(A', A)] \\
\mathsf{select}^{\mathsf{b}}(\lambda x^{\tau}.M, A) &= \lambda x^{\mathsf{select}^{\mathsf{b}}(\tau, A)}.\ \mathsf{select}^{\mathsf{b}}(M, A) \\
\mathsf{select}^{\mathsf{b}}(M\ N, A) &= (\mathsf{select}^{\mathsf{b}}(M, A))\ (\mathsf{select}^{\mathsf{b}}(N, A)) \\
\mathsf{select}^{\mathsf{b}}(x^{\tau}, A) &= x^{\mathsf{select}^{\mathsf{b}}(\tau, A)}
\end{aligned}$$

Finally, let $\mathsf{select}^{\mathsf{b}}$ be extended to environments as follows. Let $\mathsf{select}^{\mathsf{b}}(E, A)$ be defined iff $\mathsf{select}^{\mathsf{b}}(E(x), A)$ is defined for all $x \in \mathsf{dom}(E)$. In this case, let it be defined by

$$\mathsf{select}^{\mathsf{b}}(E, A)(x) = \mathsf{select}^{\mathsf{b}}(E(x), A).$$

Lemma 8.5 (Properties of $\mathsf{select}^{\mathsf{b}}$)**.**

1. *$(A \triangleleft \kappa)$ if and only if $\mathsf{select}^{\mathsf{b}}(\kappa, A)$ is defined.*
2. *If $(A \triangleleft \kappa)$ and $(\tau : \kappa)$, then $(\mathsf{select}^{\mathsf{b}}(\tau, A) : \mathsf{select}^{\mathsf{b}}(\kappa, A))$.*
3. *If $(A \triangleleft \kappa)$ and $(A' : \kappa)$, then $(\mathsf{select}^{\mathsf{b}}(A', A) : \mathsf{select}^{\mathsf{b}}(\kappa, A))$.*
4. *If $(A \triangleleft \lfloor P \rfloor)$, then $\mathsf{select}^{\mathsf{b}}(P, A)$ is defined.*
5. *If $(A \triangleleft \lfloor P \rfloor)$, then $(\lfloor \mathsf{select}^{\mathsf{b}}(P, A) \rfloor = \mathsf{select}^{\mathsf{b}}(\lfloor P \rfloor, A))$.*
6. *If $(A \triangleleft \lfloor P \rfloor)$, then $(\lceil \mathsf{select}^{\mathsf{b}}(P, A) \rceil = \mathsf{select}^{\mathsf{b}}(\lceil P \rceil, A))$.*
7. *If $(A \triangleleft \lfloor P \rfloor)$ and $(\tau : \lfloor P \rfloor)$,*
   then $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, P), A) \;\leftrightarrow_{\mathsf{ty}}\; \mathsf{expand}(\mathsf{select}^{\mathsf{b}}(\tau, A), \mathsf{select}^{\mathsf{b}}(P, A)))$.
8. *If $(A \triangleleft \lfloor P \rfloor)$ and $(\tau : \lceil P \rceil)$,*
   then $(\mathsf{select}^{\mathsf{b}}(\forall P.\tau, A) \;\leftrightarrow_{\mathsf{ty}}\; \forall(\mathsf{select}^{\mathsf{b}}(P, A)).\mathsf{select}^{\mathsf{b}}(\tau, A))$.
9. *If $(A \triangleleft \kappa)$, $(\tau : \kappa)$ and $(\mathsf{select}^{\mathsf{i}}(\tau, A') = \tau')$,*
   then $(\mathsf{select}^{\mathsf{b}}(\tau', A) \;\leftrightarrow_{\mathsf{ty}}\; \mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau, A), \mathsf{select}^{\mathsf{b}}(A', A)))$. □

Proof. Statements (1) through (6) are all proved, separately, by induction on the structure of A. Statements (7) through (9) are proved similarly to statements (8) through (10) of Lemma 8.2. □

Lemma 8.6. *If $(E \vdash^{\mathsf{B}} M : \tau \;\mathsf{at}\; \kappa)$ and $(A \triangleleft \kappa)$, then $(\mathsf{select}^{\mathsf{b}}(E, A) \vdash^{\mathsf{B}} \mathsf{select}^{\mathsf{b}}(M, A) : \mathsf{select}^{\mathsf{b}}(\tau, A) \;\mathsf{at}\; \mathsf{select}^{\mathsf{b}}(\kappa, A))$.* □

Proof. By induction on the derivation of $(E \vdash^{\mathsf{B}} M : \tau \;\mathsf{at}\; \kappa)$, using Lemma 8.5. □

9 Relating Branching Types to Intersection Types

9.1 From Branching Types to Intersection Types

Individual λ^{B}-types are closely related to types of the intersection type system λ^{I} from Section 2.2. A λ^{I}-type is obtained from an individual λ^{B}-type by first normalizing the λ^{B}-type and then erasing all type selection parameters. Formally, we define a function $|\cdot|$ that maps individual λ^{B}-types to λ^{I}-types:

$$
\begin{array}{lcl}
|\alpha| & = & \alpha \\
|\bar{\mu} \to \bar{\nu}| & = & |\bar{\mu}| \to |\bar{\nu}| \\
|\forall \mathsf{join}\{i = \bar{P}_i\}^I.\{i = \mu_i\}^I| & = & \wedge\{i = |\forall \bar{P}_i.\mu_i|\}^I \\
|\tau| & = & |\mathsf{nf}(\tau)| \quad \text{if } \neg\mathsf{normal}(\tau) \\
|E|(x) & = & |E(x)|
\end{array}
$$

We use the same symbol $|\cdot|$ to denote the *type erasure* function from λ^{B}-terms to λ^{I}-terms:

$$
\begin{array}{rcl}
|\Lambda P.M| & = & |M| \\
|M[A]| & = & |M|
\end{array}
\qquad
\begin{array}{rcl}
|\lambda x^{\tau}.M| & = & \lambda x.|M| \\
|MN| & = & |M|\;|N|
\end{array}
\qquad
|x^{\tau}| = x
$$

We will show that $(E \vdash^{\mathsf{B}} M : \tau \;\mathsf{at}\; *)$ implies that $(|E| \vdash^{\mathsf{i}} |M| : |\tau|)$, i.e., the operator $|\cdot|$ maps λ^{B}-typing judgments at branching shape $*$ to λ^{I}-typing judgments. This fact is an instance of a more general theorem about λ^{B}-judgments at arbitrary branching shapes: Judgments of the form $(E \vdash^{\mathsf{B}} M : \tau \;\mathsf{at}\; \kappa)$ are mapped to *sets* of λ^{I}-judgments—each maximal path in κ gives rise to a separate λ^{I}-judgment.

Definition 9.1 (Maximal Paths of Kinds)**.** An individual argument $\bar{A}$ is called a *maximal path* of kind κ, if $(\mathsf{select}^{\mathsf{b}}(\kappa, \bar{A}) = *)$. We write $(\bar{A} \lhd^{\mathsf{max}} \kappa)$ iff $\bar{A}$ is a maximal path of κ. $\mathsf{Maxpath}(\kappa)$ denotes the set of all maximal paths of κ. □

Lemma 9.2.

1. *If* $(\bar{A} \lhd^{\mathsf{max}} \kappa)$*, then* $(\bar{A} \lhd \kappa)$.
2. *If* $(\bar{A} \lhd^{\mathsf{max}} \lceil \bar{P} \rceil)$*, then* $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, \bar{P}), \bar{A}) \leftrightarrow_{\mathsf{ty}} \tau)$. □

Proof. (1) is proved by induction on the structure of $\bar{A}$, and (2) by induction on the structure of $\bar{P}$. □

The function $\mathsf{select}^{\mathsf{b}}$ is extended to a partial function from environments to environments as follows: $\mathsf{select}^{\mathsf{b}}(E, A)$ is defined iff $\mathsf{select}^{\mathsf{b}}(E(x), A)$ is defined for all $x \in \mathsf{dom}(E)$. In that case, it is defined by $\mathsf{select}^{\mathsf{b}}(E, A)\ (x) = \mathsf{select}^{\mathsf{b}}(E(x), A)$.

Theorem 9.3 (Soundness of $|\cdot|$)**.**
If $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa)$ *and* $(\bar{A} \lhd^{\mathsf{max}} \kappa)$*, then* $(|\mathsf{select}^{\mathsf{b}}(E, \bar{A})| \vdash^{\mathsf{i}} |M| : |\mathsf{select}^{\mathsf{b}}(\tau, \bar{A})|)$. □

Corollary 9.4. *If* $(E \vdash^{\mathsf{B}} M : \tau \text{ at } *)$*, then* $(|E| \vdash^{\mathsf{i}} |M| : |\tau|)$. □

Corollary 9.4 follows from Theorem 9.3 because $(* \lhd^{\mathsf{max}} *)$, $(\mathsf{select}^{\mathsf{b}}(E, *) \leftrightarrow_{\mathsf{ty}} E)$ and $(\mathsf{select}^{\mathsf{b}}(\tau, *) \leftrightarrow_{\mathsf{ty}} \tau)$. The remainder of this section is devoted to the proof of Theorem 9.3. The proof is by lexicographical induction on the pair $\langle \mathsf{size}(M), \mathsf{size}(\kappa) \rangle$, where size is defined as follows:

$$
\begin{array}{rcl@{\qquad}rcl}
\mathsf{size}(x^{\tau}) & = & 0; & \mathsf{size}(\lambda x^{\tau}.M) & = & \mathsf{size}(M) + 1; \\
\mathsf{size}(M\ N) & = & \mathsf{size}(M) + \mathsf{size}(N) + 1; & \mathsf{size}(\Lambda P.M) & = & \mathsf{size}(M) + 1; \\
\mathsf{size}(M[A]) & = & \mathsf{size}(M) + 1; & & &
\end{array}
$$

$$
\mathsf{size}(*) = 0; \qquad \mathsf{size}(\{i = \kappa_i\}^I) = \mathsf{Max}\{\, \mathsf{size}(\kappa_i) \mid i \in I\,\} + 1
$$

Lemma 9.5. *If* $(\mathsf{select}^{\mathsf{b}}(M, A) = N)$*, then* $(\mathsf{size}(M) = \mathsf{size}(N))$. □

Lemma 9.6. $\mathsf{select}^{\mathsf{b}}(\tau, (i, \bar{A})) \ \leftrightarrow_{\mathsf{ty}}^{\perp}\ \mathsf{select}^{\mathsf{b}}(\mathsf{select}^{\mathsf{b}}(\tau, (i, *)), \bar{A})$. □

Part (2) of the following lemma simulates the $(\forall_{\mathsf{e}})$-rule on λ^{I}-typing-judgments. Part (1) simulates an instance of the $(\forall_{\mathsf{i}})$-rule. (The simulation of this instance suffices for the proof of Theorem 9.3.)

Lemma 9.7.

1. *If* $(\tau : \lceil \bar{P} \rceil)$ *and* $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\mathsf{select}^{\mathsf{b}}(\tau, \bar{A})|)$ *for all* $\bar{A} \in \mathsf{Maxpath}(\lceil \bar{P} \rceil)$*,*
   then $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\forall \bar{P}.\tau|)$.
2. *If* $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\tau|)$ *and* $(\mathsf{select}^{\mathsf{i}}(\tau, \bar{A}) \leftrightarrow_{\mathsf{ty}} \tau')$*, then* $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\tau'|)$. □

Proof. (1): It suffices to prove the statement for normal τ. The proof is by induction on the structure of $\bar{P}$. Suppose that τ is normal, $(\tau : \lceil \bar{P} \rceil)$ and $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\mathsf{select}^{\mathsf{b}}(\tau, \bar{A})|)$ for all $\bar{A} \in \mathsf{Maxpath}(\lceil \bar{P} \rceil)$.

Case, $\bar{P} = *$**:** Because $(* \in \mathsf{Maxpath}(*))$, it is the case that $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\mathsf{select}^{\mathsf{b}}(\tau, *)|)$. But $\mathsf{select}^{\mathsf{b}}(\tau, *) \leftrightarrow_{\mathsf{ty}} \forall * .\tau$.

Case, $\bar{P} = \mathsf{join}\{i = \bar{P}_i\}^I$**:** Because τ is normal, it is of the form $\tau = \{i = \tau_i\}^I$ where $\tau_i : \lceil \bar{P}_i \rceil$ for all $i \in I$.

Claim: $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\mathsf{select}^{\mathsf{b}}(\tau_i, \bar{A})|)$ *for all* $i \in I$ *and all* $\bar{A}$ *in* $\mathsf{Maxpath}(\lceil \bar{P}_i \rceil)$. To prove this claim, suppose that $i \in I$ and $\bar{A} \in \mathsf{Maxpath}(\lceil \bar{P}_i \rceil)$. Then $(i, \bar{A}) \in \mathsf{Maxpath}(\lceil \bar{P} \rceil)$. Then $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\mathsf{select}^{\mathsf{b}}(\tau, (i, \bar{A}))|)$, by assumption. But $(\mathsf{select}^{\mathsf{b}}(\tau, (i, \bar{A})) = \mathsf{select}^{\mathsf{b}}(\tau_i, \bar{A}))$.

Now, by induction hypotheses, $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\forall \bar{P}_i.\tau_i|)$ for all $i \in I$. Then $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : \wedge\{i = |\forall \bar{P}_i.\tau_i|\}^I)$, by $(\wedge_{\mathsf{i}})$. But $\wedge\{i = |\forall \bar{P}_i.\tau_i|\}^I = |\forall \bar{P}.\tau|$.

(2): By induction on the structure of $\bar{A}$. Suppose $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\tau|)$ and $(\mathsf{select}^{\mathsf{i}}(\tau, \bar{A}) \leftrightarrow_{\mathsf{ty}} \tau')$.

Case, $\bar{A} = *$: Then $(\tau' \leftrightarrow_{\mathsf{ty}} \mathsf{select}^{\mathsf{i}}(\tau, \bar{A}) = \mathsf{select}^{\mathsf{i}}(\tau, *) \leftrightarrow_{\mathsf{ty}} \tau)$. Then $|\tau'| = |\tau|$. Then $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\tau'|)$, because $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\tau|)$

Case, $\bar{A} = (j, \bar{A}')$: Then there are a set I, and families $(\bar{P}_i)_{i \in I}$ and $(\tau_i)_{i \in I}$ such that $(\mathsf{nf}(\tau) = \forall \mathsf{join}\{i = \bar{P}_i\}^I.\{i = \tau_i\}^I)$, $(j \in I)$ and $(\mathsf{select}^{\mathsf{i}}(\forall \bar{P}_j.\tau_j, \bar{A}') \leftrightarrow_{\mathsf{ty}} \tau')$. Moreover, $(|\tau| = \wedge\{i = |\forall \bar{P}_i.\tau_i|\}^I)$, by definition of $|\cdot|$. Then $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\forall \bar{P}_j.\tau_j|)$, by $(\wedge_{\mathsf{e}})$. Then $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : |\tau'|)$, by induction hypotheses. □

Proof of Theorem 9.3. The proof is by lexicographical induction on the pair $\langle \mathsf{size}(M), \mathsf{size}(\kappa) \rangle$. Suppose $(E \vdash^{\mathsf{B}} M : \tau \;\mathsf{at}\; \kappa)$ and $(\bar{A} \lhd^{\mathsf{max}} \kappa)$.

Case, $\kappa = \{i = \kappa_i\}^I$: By definition of $\lhd^{\mathsf{max}}$, there are $\bar{A}'$ and i in I such that $(\kappa = (i, \bar{A}'))$ and $(\bar{A}' \lhd^{\mathsf{max}} \kappa_i)$. By Lemmas 9.2(1) and 8.6,

$$\mathsf{select}^{\mathsf{b}}(E, (i, *)) \vdash^{\mathsf{B}} \mathsf{select}^{\mathsf{b}}(M, (i, *)) : \mathsf{select}^{\mathsf{b}}(\tau, (i, *)) \;\mathsf{at}\; \kappa_i$$

By Lemma 9.5, $(\mathsf{size}(\mathsf{select}^{\mathsf{b}}(M, (i, *))) = \mathsf{size}(M))$. Therefore, by induction hypothesis and Lemma 10.9,

$$|\mathsf{select}^{\mathsf{b}}(\mathsf{select}^{\mathsf{b}}(E, (i, *)), \bar{A}')| \vdash^{\mathsf{i}} |M| : |\mathsf{select}^{\mathsf{b}}(\mathsf{select}^{\mathsf{b}}(\tau, (i, *)), \bar{A}')|$$

Then, by Lemma 9.6,

$$|\mathsf{select}^{\mathsf{b}}(E, \bar{A})| \vdash^{\mathsf{i}} |M| : |\mathsf{select}^{\mathsf{b}}(\tau, \bar{A})|$$

Case, $\kappa = *$: Because $(\bar{A} \lhd^{\mathsf{max}} *)$, it is the case that $(\bar{A} = *)$. Because $(\mathsf{select}^{\mathsf{b}}(E, *) = E)$ and $(\mathsf{select}^{\mathsf{b}}(\tau, *) = \tau)$, we have to show that $(|E| \vdash^{\mathsf{i}} |M| : |\tau|)$. We distinguish cases by the outermost term constructor of M. The cases where M is a variable, λ-abstraction or application are straightforward.

Subcase, $M = \Lambda \bar{P}.N$: By inversion of the typing rule for Λ, there is a type σ such that $(\tau \leftrightarrow_{\mathsf{ty}} \forall \bar{P}.\sigma)$ and $(\mathsf{expand}(E, \bar{P}) \vdash^{\mathsf{B}} N : \sigma \;\mathsf{at}\; \lceil \bar{P} \rceil)$. By induction hypothesis:

$$(\forall \bar{A} \in \mathsf{Maxpath}(\lceil \bar{P} \rceil))\,(|\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(E, \bar{P}), \bar{A})| \vdash^{\mathsf{i}} |N| : |\mathsf{select}^{\mathsf{b}}(\sigma, \bar{A})|)$$

Then, by Lemma 9.2(2):

$$(\forall \bar{A} \in \mathsf{Maxpath}(\lceil \bar{P} \rceil))\,(|E| \vdash^{\mathsf{i}} |N| : |\mathsf{select}^{\mathsf{b}}(\sigma, \bar{A})|)$$

Then $(|E| \vdash^{\mathsf{i}} |M| : |\tau|)$, by Lemma 9.7(1).

Subcase, $M = N[\bar{A}]$: By inversion of the typing rule for type selection application, there is a type σ such that $(\mathsf{select}^{\mathsf{i}}(\sigma, \bar{A}) \leftrightarrow_{\mathsf{ty}} \tau)$ and $(E \vdash^{\mathsf{B}} N : \sigma \;\mathsf{at}\; *)$. Then, by induction hypothesis, $(|E| \vdash^{\mathsf{i}} |N| : |\sigma|)$. Then $(|E| \vdash^{\mathsf{i}} |M| : |\tau|)$, by Lemma 9.7(2). □

9.2 From Intersection Types to Branching Types

We define the following mapping from λ^{I}-types to individual λ^{B}-types:

$$\mathsf{e}(\alpha) = \alpha; \quad \mathsf{e}(\hat{\tau} \to \hat{\sigma}) = \mathsf{e}(\hat{\tau}) \to \mathsf{e}(\hat{\sigma}); \quad \mathsf{e}(\wedge\{i = \hat{\tau}_i\}^I) = \forall \mathsf{join}\{i = *\}^I.\{i = \mathsf{e}(\tau_i)\}^I;$$
$$\mathsf{e}(\hat{E})(x) = \mathsf{e}(\hat{E}(x))$$

Lemma 9.8. $\mathsf{e}(\hat{\tau}) : *$ □

Proof. By induction on the structure of $\hat{\tau}$. □

The mapping e is a right inverse of $|\cdot|$, i.e., $|\mathsf{e}(\hat{\tau})| = \hat{\tau}$ for all λ^{I}-types $\hat{\tau}$. Moreover, it is the case that $\mathsf{e}(|\tau|) = \mathsf{nf}(\tau)$ for all *individual* λ^{B}-types τ. Given $\hat{M}$ such that $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : \hat{\tau})$, we will show how to construct a λ^{B}-term M such that $|M| = \hat{M}$ and $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} M : \mathsf{e}(\hat{\tau}) \;\mathsf{at}\; *)$. This construction is described in the proof of Theorem 9.9. It is not hard to extract from this proof an algorithm for translating well-typed λ^{CIL}-terms to λ^{B}-terms of corresponding types.

Theorem 9.9. *If* $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : \hat{\tau})$*, then there exists a* λ^{B}*-term* M *such that* $|M| = \hat{M}$ *and* $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} M : \mathsf{e}(\hat{\tau})$ at $*)$. □

To prepare the proof of this theorem, we first extend the branching operator $\{i = \ldots\}^I$ from types to environments in the obvious way:

$$\{i = E_i\}^I(x) \quad = \quad \{i = E_i(x)\}^I$$

The following Lemma 9.10 states that the branching operator can also be extended to a term operator $(M_i)_I \mapsto \{i = M_i\}^I$, provided that the M_i all have the same type erasure. This fact is key for our translation from λ^{I} to λ^{B}. If we interpret the constructive proof of Theorem 9.9 as an algorithm that converts λ^{CIL}-terms to λ^{B}-terms, then Lemma 9.10 corresponds to a subprocedure that is called to merge a recursively computed family of λ^{B}-terms $(M_i)_I$ into a single λ^{B}-term corresponding to a virtual tuple in λ^{CIL}.

Lemma 9.10 (Branching Terms)**.** *If* $|M_i| = \hat{M}$ *and* $(E_i \vdash^{\mathsf{B}} M_i : \tau_i$ at $\kappa_i)$ *for all* i *in* I*, then there exists a* λ^{B}*-term* $\{i = M_i\}^I$ *such that* $|\{i = M_i\}^I| = \hat{M}$ *and* $(\{i = E_i\}^I \vdash^{\mathsf{B}} \{i = M_i\}^I : \{i = \tau_i\}^I$ at $\{i = \kappa_i\}^I)$. □

Hidden in the proof of Lemma 9.10 is an algorithm for computing the mapping $(M_i)_I \mapsto \{i = M_i\}^I$. It is instructive to see what the algorithm computes on an example input, before delving into the proof of Lemma 9.10:

Example 9.11. For brevity, we omit redundant type annotations on non-binding variable occurrences. Consider the following λ^{B}-terms:

$$\begin{array}{rcl} M_l & = & \Lambda \mathsf{join}\{i = *, j = *\}\,.\,\lambda x^{\{i=\alpha, j=\beta\}}\,.\,x \\ M_m & = & \lambda x^{\forall \mathsf{join}\{k=*,h=*\}.\{k=\gamma,h=\delta\}}\,.\,x[(k,*)] \end{array}$$

These terms have equal type erasures: $|M_l| = |M_m| = \lambda x.x$. They are both well-typed:

$$\begin{array}{ll} \vdash^{\mathsf{B}} M_l \,:\, \tau_l \text{ at } * & \text{where } \tau_l = \forall \mathsf{join}\{i = *, j = *\}\,.\{i = \alpha \to \alpha, j = \beta \to \beta\} \\ \vdash^{\mathsf{B}} M_m \,:\, \tau_m \text{ at } * & \text{where } \tau_m = (\forall \mathsf{join}\{k = *, h = *\}\,.\,\{k = \gamma, h = \delta\}) \to \gamma \end{array}$$

The proof of Lemma 9.10 constructs the following "branching term" $\{l = M_l, m = M_m\}$:

$$\begin{array}{rcl} \{l = M_l, m = M_m\} & = & \Lambda\{l = \mathsf{join}\{i = *, j = *\}, m = *\}\,. \\ & & \quad \lambda x^{\{l=\{i=\alpha,j=\beta\},\ m=\forall \mathsf{join}\{k=*,\ h=*\}.\{k=\gamma,h=\delta\}\}}\,. \\ & & \qquad x[l = \{i = *, j = *\},\ m = (k, *)] \end{array}$$

It is the case that $|\{l = M_l, m = M_m\}| = \lambda x.x$. The reader is invited to convince herself that $\vdash^{\mathsf{B}} \{l = M_l, m = M_m\} \,:\, \{l = \tau_l, m = \tau_m\}$ at $\{l = *, m = *\}$. □

Proof of Lemma 9.10. By induction on $\sum_{i \in I} f(M_i)$, where $f(M_i)$ is the height of M_i. Let M be an untyped term and $(M_i)_{i \in I}$ a family of λ^{B}-terms such that $|M_i| = \hat{M}$ and $(E_i \vdash^{\mathsf{B}} M_i : \tau_i$ at $\kappa_i)$ for all i in I.

Case, there exists $j \in I$ such that M_j is of the form $M_j = \Lambda P_j.M'_j$: By inverting $(E_j \vdash^{\mathsf{B}} M_j : \tau_j$ at $\kappa_j)$, there exists τ'_j such that $\tau_j \leftrightarrow_{\mathsf{ty}} \forall P_j.\tau'_j$, $\kappa_j = \lfloor P_j \rfloor$ and $(\mathsf{expand}(E_j, P_j) \vdash^{\mathsf{B}} M'_j : \tau'_j$ at $\lceil P_j \rceil)$. Let $E' = \{i = E_i\}^{I-\{j\}} \cup \{j = \mathsf{expand}(E_j, P_j)\}$, $M' = \{i = M_i\}^{I-\{j\}} \cup \{j = M'_j\}$, $\tau' = \{i = \tau_i\}^{I-\{j\}} \cup \{j = \tau'_j\}$ and $\kappa' = \{i = \kappa_i\}^{I-\{j\}} \cup \{j = \lceil P_j \rceil\}$. By induction hypothesis, $|M'| = \hat{M}$ and $(E' \vdash^{\mathsf{B}} M' : \tau'$ at $\kappa')$. Let $P' = \{i = \kappa_i\}^{I-\{j\}} \cup \{j = P_j\}$. The following equations hold:

$$\begin{array}{rcl} \mathsf{expand}(\{i = E_i\}^I, P') & \leftrightarrow_{\mathsf{ty}} & \{i = \mathsf{expand}(E_i, \kappa_i)\}^{I-\{j\}} \cup \{j = \mathsf{expand}(E_j, P_j)\} \\ & \leftrightarrow_{\mathsf{ty}} & \{i = E_i\}^{I-\{j\}} \cup \{j = \mathsf{expand}(E_j, P_j)\} \\ & \leftrightarrow_{\mathsf{ty}} & E' \end{array}$$

Moreover, $\lceil P' \rceil = \kappa'$ and $\lfloor P' \rfloor = \kappa$. Therefore, $(\{i = E_i\}^I \vdash^{\mathsf{B}} \Lambda P'.M' : \forall P'.\tau' \text{ at } \{i = \kappa_i\}^I)$, by $(\forall_{\mathsf{i}})$.

$$\begin{array}{lll} \forall P'.\tau' & \leftrightarrow_{\mathsf{ty}} & \{i = \forall \kappa_i.\tau_i\}^{I-\{j\}} \cup \{j = \forall P_j.\tau'_j\} \\ & \leftrightarrow_{\mathsf{ty}} & \{i = \tau_i\}^I \end{array}$$

It follows that $(\{i = E_i\}^I \vdash^{\mathsf{B}} \Lambda P'.M' : \{i = \tau_i\}^I \text{ at } \{i = \kappa_i\}^I)$. Now, define $\{i = M_i\}^I$ to be $\Lambda P'.M'$. Then $|\{i = M_i\}^I| = |M'| = \hat{M}$, and $\{i = M_i\}^I$ has the desired type.

Case, there exists $j \in I$ such that M_j is of the form $M_j = M'_j[A_j]$: By inverting $(E_j \vdash^{\mathsf{B}} M_j : \tau_j \text{ at } \kappa_j)$, there exists τ'_j such that $\tau_j \leftrightarrow_{\mathsf{ty}} \mathsf{select}^{\mathsf{i}}(\tau'_j, A_j)$ and $(E_j \vdash^{\mathsf{B}} M'_j : \tau'_j \text{ at } \kappa_j)$. Let $M' = \{i = M_i\}^{I-\{j\}} \cup \{j = M'_j\}$ and $\tau' = \{i = \tau_i\}^{I-\{j\}} \cup \{j = \tau'_j\}$. By induction hypothesis, $|M'| = \hat{M}$ and $(\{i = E_i\}^I \vdash^{\mathsf{B}} M' : \tau' \text{ at } \{i = \kappa_i\}^I)$. Let $A' = \{i = \kappa_i\}^{I-\{j\}} \cup \{j = A_j\}$. The following equations hold:

$$\begin{array}{lll} \mathsf{select}^{\mathsf{i}}(\tau', A') & \leftrightarrow_{\mathsf{ty}} & \{i = \mathsf{select}^{\mathsf{i}}(\tau_i, \kappa_i)\}^{I-\{j\}} \cup \{j = \mathsf{select}^{\mathsf{i}}(\tau'_j, A_j)\} \\ & \leftrightarrow_{\mathsf{ty}} & \{i = \tau_i\}^I \end{array}$$

Therefore, $(\{i = E_i\}^I \vdash^{\mathsf{B}} M'[A'] : \{i = \tau_i\}^I \text{ at } \{i = \kappa_i\}^I)$, by $(\forall_{\mathsf{e}})$. Now, define $\{i = M_i\}^I$ to be $M'[A']$. Then $|\{i = M_i\}^I| = |M'| = \hat{M}$, and $\{i = M_i\}^I$ has the desired type.

From this point on, we assume that none of the previous two cases apply. Because $|M_i| = M$ for all i in I, it then follows that either all M_i are the same variable, or they are all λ-abstractions, or they are all applications.

Case, $\hat{M} = x$: Then M_i is of the form $M_i = x^{\sigma_i}$ for all i in I. By inverting $(E_i \vdash^{\mathsf{B}} M_i : \tau_i \text{ at } \kappa_i)$, it is the case that $E_i(x) \leftrightarrow_{\mathsf{ty}} \sigma_i \leftrightarrow_{\mathsf{ty}} \tau_i$ for all i in I. Let $\sigma = \{i = \sigma_i\}^I$ and $M' = x^\sigma$. Then $\{i = E_i\}(x) \leftrightarrow_{\mathsf{ty}} \sigma$. Therefore, $(\{i = E_i\}^I \vdash^{\mathsf{B}} M' : \sigma \text{ at } \{i = \kappa_i\}^I)$, by (ax). Because $\sigma \leftrightarrow_{\mathsf{ty}} \{i = \tau_i\}^I$, it follows that $(\{i = E_i\}^I \vdash^{\mathsf{B}} M' : \{i = \tau_i\}^I \text{ at } \{i = \kappa_i\}^I)$. Now, define $\{i = M_i\}^I$ to be M'. Then $|\{i = M_i\}^I| = |x^\sigma| = x = \hat{M}$, and $\{i = M_i\}^I$ has the desired type.

Case, $\hat{M} = \lambda x.\hat{N}$: Then M_i is of the form $\lambda x^{\sigma_i}.N'_i$ for all i in I. By inverting $(E_i \vdash^{\mathsf{B}} M_i : \tau_i \text{ at } \kappa_i)$, it is the case that there exists σ'_i such that $\tau_i \leftrightarrow_{\mathsf{ty}} \sigma_i \to \sigma'_i$ and $(E_i[x_i \mapsto \sigma_i] \vdash^{\mathsf{B}} N'_i : \sigma'_i \text{ at } \kappa_i)$ for all i in I. Let $E' = \{i = E_i[x \mapsto \sigma_i]\}^I$, $N' = \{i = N'_i\}^I$, $\sigma' = \{i = \sigma'_i\}^I$ and $\kappa = \{i = \kappa_i\}^I$. By induction hypothesis, $|N'| = \hat{N}$ and $(E' \vdash^{\mathsf{B}} N' : \sigma' \text{ at } \kappa)$. Let $\sigma = \{i = \sigma_i\}^I$. Then $E' = \{i = E_i\}^I[x \mapsto \sigma]$. Therefore, $(\{i = E_i\}^I \vdash^{\mathsf{B}} \lambda x^\sigma.N' : \sigma \to \sigma' \text{ at } \kappa)$, by $(\to_{\mathsf{i}})$. Because $\sigma \to \sigma' \leftrightarrow_{\mathsf{ty}} \{i = \tau_i\}^I$, it follows that $(\{i = E_i\}^I \vdash^{\mathsf{B}} \lambda x^\sigma.N' : \{i = \tau_i\}^I \text{ at } \kappa)$. Now, define $\{i = M_i\}^I$ to be $\lambda x^\sigma.N'$. Then $|\{i = M_i\}^I| = \lambda x.|N'| = \lambda x.\hat{N} = \hat{M}$, and $\{i = M_i\}^I$ has the desired type.

Case, $\hat{M} = (\hat{N}_1\ \hat{N}_2)$: Then M_i is of the form $(N_{1,i}\ N_{2,i})$ for all i in I. By inverting $(E_i \vdash^{\mathsf{B}} M_i : \tau_i \text{ at } \kappa_i)$, it is the case that there exist σ_i, σ'_i such that $\tau_i \leftrightarrow_{\mathsf{ty}} \sigma_i$, $(E_i \vdash^{\mathsf{B}} N_{1,i} : \sigma'_i \to \sigma_i \text{ at } \kappa_i)$ and $(E_i \vdash^{\mathsf{B}} N_{2,i} : \sigma'_i \text{ at } \kappa_i)$ for all i in I. Let $E = \{i = E_i\}^I$, $N'_1 = \{i = N'_{1,i}\}^I$, $N'_2 = \{i = N'_{2,i}\}^I$, $\sigma = \{i = \sigma_i\}^I$, $\sigma' = \{i = \sigma'_i\}^I$ and $\kappa = \{i = \kappa_i\}^I$. By induction hypothesis, $|N'_1| = \hat{N}_1$, $|N'_2| = \hat{N}_2$, $(E \vdash^{\mathsf{B}} N'_1 : \{i = \sigma'_i \to \sigma_i\}^I \text{ at } \kappa)$ and $(E \vdash^{\mathsf{B}} N'_2 : \sigma' \text{ at } \kappa)$. Because $\{i = \sigma'_i \to \sigma_i\} \leftrightarrow_{\mathsf{ty}} \sigma' \to \sigma$, it follows that $(E \vdash^{\mathsf{B}} N'_1 : \sigma' \to \sigma \text{ at } \kappa)$. Then $(E \vdash^{\mathsf{B}} N'_1\ N'_2 : \sigma \text{ at } \kappa)$, by $(\to_{\mathsf{e}})$. Now, define $\{i = M_i\}^I$ to be $(N'_1\ N'_2)$. Then $|\{i = M_i\}^I| = |N'_1|\ |N'_2| = (\hat{N}_1\ \hat{N}_2) = M$, and $\{i = M_i\}^I$ has the desired type. □

Proof of Theorem 9.9. By induction on the derivation of $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : \hat{\tau})$. Suppose $\mathcal{D}$ is a derivation of $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : \hat{\tau})$.

Case, $\mathcal{D}$ ends in (ax): Then $\hat{M} = x$ and $\hat{\tau} = E(x)$ for some x. By lemma 9.8, $(\mathsf{e}(\hat{E}) : *)$. Moreover, $\mathsf{e}(\hat{\tau}) \leftrightarrow_{\mathsf{ty}} \mathsf{e}(\hat{E})(x)$, by reflexivity. Therefore, $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} x^{\mathsf{e}(\hat{\tau})} : \mathsf{e}(\hat{\tau}) \text{ at } *)$, by (ax).

Case, $\mathcal{D}$ ends in $(\to_{\mathsf{i}})$: Then $\hat{M} = \lambda x.\hat{N}$, $\hat{\tau} = \hat{\tau}_1 \to \hat{\tau}_2$ and $(\hat{E}[x \mapsto \hat{\tau}_1] \vdash^{\mathsf{i}} \hat{N} : \hat{\tau}_2)$ for some $x, \hat{N}, \hat{\tau}_1, \hat{\tau}_2$. By induction hypothesis, there exists a λ^{B}-term N such that $|N| = \hat{N}$ and $(\mathsf{e}(\hat{E})[x \mapsto \mathsf{e}(\hat{\tau}_1)] \vdash^{\mathsf{B}} N : \mathsf{e}(\hat{\tau}_2) \text{ at } *)$. Let $M = \lambda x^{\mathsf{e}(\hat{\tau}_1)}.N$. Then $|M| = \hat{M}$ and $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} M : \mathsf{e}(\hat{\tau}) \text{ at } *)$, by $(\to_{\mathsf{i}})$.

Case, $\mathcal{D}$ ends in $(\rightarrow_{\mathsf{e}})$: Then $\hat{M} = (\hat{M}_1\ \hat{M}_2)$, $(\hat{E} \vdash^{\mathsf{i}} \hat{M}_1 : \hat{\sigma} \rightarrow \hat{\tau})$ and $(\hat{E} \vdash^{\mathsf{i}} \hat{M}_2 : \hat{\sigma})$ for some $\hat{M}_1, \hat{M}_2, \hat{\sigma}$. By induction hypothesis, there exist λ^{B}-terms M_1 and M_2 such that $|M_1| = \hat{M}_1$, $|M_2| = \hat{M}_2$, $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} M_1 : \mathsf{e}(\hat{\sigma}) \rightarrow \mathsf{e}(\hat{\tau})\ \mathsf{at}\ *)$ and $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} M_2 : \mathsf{e}(\hat{\sigma})\ \mathsf{at}\ *)$. Let $M = M_1\ M_2$. Then $|M| = \hat{M}$ and $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} M : \mathsf{e}(\hat{\tau})\ \mathsf{at}\ *)$, by $(\rightarrow_{\mathsf{e}})$.

Case, $\mathcal{D}$ ends in $(\wedge_{\mathsf{i}})$: Then $\hat{\tau}$ is of the form $\hat{\tau} = \wedge\{i = \hat{\tau}_i\}_I$ and $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : \hat{\tau}_i)$ for all i in I. By induction hypothesis, there exist λ^{B}-terms M_i such that $|M_i| = \hat{M}$ and $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} M_i : \mathsf{e}(\hat{\tau}_i)\ \mathsf{at}\ *)$ for all i in I. Let $E = \{i = \mathsf{e}(\hat{E})\}^I$, $\tau = \{i = \mathsf{e}(\hat{\tau}_i)\}^I$ and $\kappa = \{i = *\}^I$. By lemma 9.10, there exists a λ^{B}-term $\{i = M_i\}^I$ such that $|\{i = M_i\}^I| = \hat{M}$ and $(E \vdash^{\mathsf{B}} \{i = M_i\}^I : \tau\ \mathsf{at}\ \kappa)$. Let $P = \mathsf{join}(\kappa)$. Then $E = \mathsf{expand}(\mathsf{e}(\hat{E}), P)$, $\lceil P \rceil = \kappa$, $\mathsf{e}(\hat{\tau}) = \forall P.\tau$ and $\lfloor P \rfloor = *$. Therefore, $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} \Lambda P.\{i = M_i\}^I : \mathsf{e}(\hat{\tau})\ \mathsf{at}\ *)$. Let $M = \Lambda P.\{i = M_i\}^I$. Then $|M| = |\{i = M_i\}^I| = \hat{M}$, and M has the desired type.

Case, $\mathcal{D}$ ends in $(\wedge_{\mathsf{e}})$: Then there exists a type $\wedge\{i = \hat{\tau}_i\}^I$ and a j in I such that $\hat{\tau} = \hat{\tau}_j$ and $(\hat{E} \vdash^{\mathsf{i}} \hat{M} : \wedge\{i = \hat{\tau}_i\}^I)$. Let $\tau = \forall \mathsf{join}\{i = *\}^I.\{i = \mathsf{e}(\hat{\tau}_i)\}^I$. Then $\mathsf{e}(\wedge\{i = \hat{\tau}_i\}^I) = \tau$, by definition of e. By induction hypothesis, there exists a λ^{B}-term N such that $|N| = M$ and $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} N : \tau\ \mathsf{at}\ *)$. Let $\bar{A} = (j, *)$. Then $\mathsf{select}^{\mathsf{i}}(\tau, \bar{A}) \leftrightarrow_{\mathsf{ty}} \mathsf{select}^{\mathsf{i}}(\forall *.\mathsf{e}(\hat{\tau}_j), *) \leftrightarrow_{\mathsf{ty}} \mathsf{e}(\hat{\tau}_j)$. Therefore, $(\mathsf{e}(\hat{E}) \vdash^{\mathsf{B}} N[\bar{A}] : \mathsf{e}(\hat{\tau}_j)\ \mathsf{at}\ *)$, by $(\forall_{\mathsf{e}})$. Let $M = N[\bar{A}]$. Then $|M| = |N| = \hat{M}$, and M has the desired type. □

10 Term Reduction in λ^{B}

10.1 Substitution

Let a *substitution* be a finite function from Var to Term. Let s range over substitutions. Let the operation expand be extended to act as a partial function on substitutions as follows. Let $\mathsf{expand}(s, P)$ be defined iff $\mathsf{expand}(s(x), P)$ is defined for all $x \in \mathsf{dom}(s)$. In this case, let it be defined by:

$$\mathsf{expand}(s, P)(x) \quad = \quad \mathsf{expand}(s(x), P)$$

In order to define β-reduction, we define the application of substitutions to terms. We want that the type system satisfies the usual substitutivity property, as stated in Lemma 10.3. To this end, it is necessary that we expand a term N by P when substituting it for a variable x in $\Lambda P.M$. This expansion happens in the clause for $s(\Lambda P.M)$ in the inductive definition below. All other clauses of this definition are as usual. Let the *application of a substitution* s to a term M be defined inductively by the following equations.

$$\begin{array}{rcll}
s(\Lambda P.M) & = & \Lambda P.(\mathsf{expand}(s, P)(M)) & \\
s(M[A]) & = & (s(M))[A] & \\
s(\lambda x^{\tau}.M) & = & \lambda x^{\tau}.(s(M)), & \text{if } x \text{ does not occur freely in } \mathsf{ran}(s) \text{ and } x \notin \mathsf{dom}(s) \\
s(M\ N) & = & s(M)\ s(N) & \\
s(x^{\tau}) & = & s(x), & \text{if } x \in \mathsf{dom}(s) \\
s(x^{\tau}) & = & x^{\tau}, & \text{if } x \notin \mathsf{dom}(s)
\end{array}$$

Let $M[x :=^{\mathsf{b}} N]$ be the term that results from applying the singleton substitution $\{(x, N)\}$ to M.

Example 10.1. Let N, P, P', τ and σ be as in Example 8.1. Consider the term

$$M \quad = \quad \Lambda P'.\lambda x^{\tau'}.\, y^{\sigma'}\ [A]\ x^{\tau'}$$

where $\tau' = \{j = \alpha,\ l = \beta\}$, $\sigma' = \{j = \sigma,\ l = \sigma\}$ and $A = \{j = (i, *),\ l = (h, *)\}$. This term is well typed in the type environment $\{(y, \sigma)\}$. Its type erasure (defined in Section 10.4) is

$(\lambda x.\, y\, x)$. Substituting N for y in M results in

$$\begin{array}{rcl} M[y :=^{\mathsf{b}} N] & = & \Lambda P'.((\lambda x^{\tau'}.\, y^{\sigma}\ [A]\ x^{\tau'})[y :=^{\mathsf{b}} \mathsf{expand}(N, P')]) \\ & = & \Lambda P'.\lambda x^{\tau'}.\,(\mathsf{expand}(N, P'))\ [A]\ x^{\tau'} \\ & = & \Lambda P'.\lambda x^{\tau'}.\,(\Lambda\{j = P,\ l = P\}.\lambda x^{\{j=\tau,\ l=\tau\}}.x^{\{j=\tau,\ l=\tau\}})\ [A]\ x^{\tau'} \end{array}$$

□

Let typing judgments for substitutions be defined as follows:

$$(E' \vdash^{\mathsf{B}} s : E \text{ at } \kappa) \quad \overset{\text{def}}{\iff} \quad (E' \vdash^{\mathsf{B}} s(x^{E(x)}) : E(x) \text{ at } \kappa) \text{ for all } x \in \mathsf{dom}(E)$$

Lemma 10.2. *If* $(E' \vdash^{\mathsf{B}} s : E \text{ at } \lfloor P \rfloor)$,
then $(\mathsf{expand}(E', P) \vdash^{\mathsf{B}} \mathsf{expand}(s, P) : \mathsf{expand}(E, P) \text{ at } \lceil P \rceil)$. □

Proof. This follows from Corollary 8.4. □

Lemma 10.3 (Substitutivity)**.** *If* $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa)$ *and* $(E' \vdash^{\mathsf{B}} s : E \text{ at } \kappa)$*, then* $(E' \vdash^{\mathsf{B}} s(M) : \tau \text{ at } \kappa)$. □

Proof. By induction on the derivation of $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa)$, using Lemma 10.2. □

10.2 Matching Type Selection Parameters and Arguments

We define a partial function match from Parameter × Argument to Parameter × Argument × Argument. This function will be needed for the rewriting rule (Select) for type selection. The (Select)-rewriting rule will apply to redexes of the form $(\Lambda P.M)[A]$. The $\mathsf{match}(P, A)$ operation attempts to match the argument A against the type selection parameter P. It splits A into two parts — the $\mathsf{select}^{\mathsf{b}}$-*argument* A^{s} that will be fed to $\mathsf{select}^{\mathsf{b}}(M, \cdot)$, and the *remainder* A^{r} that will be pushed below the Λ. The partial function match is defined inductively by the following rules. In the second rule, note that $\lceil P \rceil$ is a trivial type selection argument for all type selection parameters P.

$$\frac{}{\mathsf{match}(*, \bar{A}) = (*, *, \bar{A})} \qquad\qquad \frac{}{\mathsf{match}(\bar{P}, *) = (\bar{P}, *, \lceil \bar{P} \rceil)}$$

$$\frac{\mathsf{match}(\bar{P}_j, \bar{A}) = (\bar{P}', \bar{A}^{\mathsf{s}}, A^{\mathsf{r}})}{\mathsf{match}(\mathsf{join}\{i = \bar{P}_i\}^I, (j, \bar{A})) = (\bar{P}', (j, \bar{A}^{\mathsf{s}}), A^{\mathsf{r}})} \quad \text{if } j \in I$$

$$\frac{\mathsf{match}(P_i, A_i) = (P_i', A_i^{\mathsf{s}}, A_i^{\mathsf{r}}) \text{ for all } i \in I}{\mathsf{match}(\{i = P_i\}^I, \{i = A_i\}^I) = (\{i = P_i'\}^I, \{i = A_i^{\mathsf{s}}\}^I, \{i = A_i^{\mathsf{r}}\}^I)}$$

Lemma 10.4 (Properties of match)**.** *Let* $\mathsf{match}(P, A) = (P', A^{\mathsf{s}}, A^{\mathsf{r}})$*. Then:*

1. $(\lfloor P \rfloor = \lfloor P' \rfloor)$.
2. $(\lceil P' \rceil = \mathsf{select}^{\mathsf{b}}(\lceil P \rceil, A^{\mathsf{s}}))$.
3. $(A^{\mathsf{r}} : \lceil P' \rceil)$.
4. $(A^{\mathsf{s}} \triangleleft \lceil P \rceil)$.
5. *If* $(\tau : \lceil P \rceil)$*, then* $(\mathsf{select}^{\mathsf{i}}(\forall P.\tau, A) \ \leftrightarrow^{\perp}_{\mathsf{ty}}\ \forall P'.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau, A^{\mathsf{s}}), A^{\mathsf{r}}))$.
6. *If* $(\tau : \lfloor P \rfloor)$*, then* $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, P), A^{\mathsf{s}}) \ \leftrightarrow_{\mathsf{ty}}\ \mathsf{expand}(\tau, P'))$. □

Proof. Statements (1) through (4) are proved, separately, by induction on the derivation of $(\mathsf{match}(P, A) = (P', A^{\mathsf{s}}, A^{\mathsf{r}}))$.

(5): By induction on the derivation of $(\mathsf{match}(P, A) = (P', A^{\mathsf{s}}, A^{\mathsf{r}}))$.

Case.

$$\frac{}{\mathsf{match}(*, \bar{A}) = (*, *, \bar{A})}$$

Suppose $(\tau : *)$.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{i}}(\forall *.\tau, \bar{A}) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{i}}(\tau, \bar{A}) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau, *), \bar{A}) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \forall *.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau, *), \bar{A}) \end{array}$$

The first equation holds by definition of $\mathsf{select}^{\mathsf{i}}$, the second one by definition of $\mathsf{select}^{\mathsf{b}}$ and the third one by definition of $\leftrightarrow_{\mathsf{ty}}$.

Case.

$$\frac{}{\mathsf{match}(\bar{P}, *) = (\bar{P}, *, \lceil \bar{P} \rceil)}$$

Suppose $(\tau : \lceil \bar{P} \rceil)$.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{i}}(\forall \bar{P}.\tau, *) \\ \leftrightarrow_{\mathsf{ty}} & \forall \bar{P}.\tau \\ \leftrightarrow_{\mathsf{ty}} & \forall \bar{P}.\mathsf{select}^{\mathsf{i}}(\tau, \lceil \bar{P} \rceil) \\ \leftrightarrow_{\mathsf{ty}} & \forall \bar{P}.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau, *), \lceil \bar{P} \rceil) \end{array}$$

The first equation holds by definition of $\mathsf{select}^{\mathsf{i}}$ and the third one by definition of $\mathsf{select}^{\mathsf{b}}$. The second equation holds by Lemma 7.24, because $\lceil \bar{P} \rceil$ is a trivial argument.

Case.

$$\frac{\mathsf{match}(\bar{P}_j, \bar{A}) = (\bar{P}', \bar{A}^{\mathsf{s}}, A^{\mathsf{r}})}{\mathsf{match}(\mathsf{join}\{i = \bar{P}_i\}^I, (j, \bar{A})) = (\bar{P}', (j, \bar{A}^{\mathsf{s}}), A^{\mathsf{r}})} \quad \text{if } j \in I$$

Let $(P = \mathsf{join}\{i = \bar{P}_i\}^I)$. Suppose $(\tau : \lceil P \rceil)$. Then τ is such that $\tau \leftrightarrow_{\mathsf{ty}} \{i = \tau_i\}^I$ where $(\tau_i : \lceil \bar{P}_i \rceil)$ for all $i \in I$.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{i}}(\forall P.\tau, (j, \bar{A})) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{i}}(\forall \bar{P}_j.\tau_j, \bar{A}) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \forall \bar{P}'.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau_j, \bar{A}^{\mathsf{s}}), A^{\mathsf{r}}) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \forall \bar{P}'.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau, (j, \bar{A}^{\mathsf{s}})), A^{\mathsf{r}}) \end{array}$$

The first and third equation hold by Lemma 7.23, and the second one by induction hypotheses.

Case.

$$\frac{\mathsf{match}(P_i, A_i) = (P'_i, A^{\mathsf{s}}_i, A^{\mathsf{r}}_i) \quad \text{for all } i \in I}{\mathsf{match}(\{i = P_i\}^I, \{i = A_i\}^I) = (\{i = P'_i\}^I, \{i = A^{\mathsf{s}}_i\}^I, \{i = A^{\mathsf{r}}_i\}^I)}$$

Let $(P = \{i = P_i\}^I)$ and $(P' = \{i = P'_i\}^I)$. Suppose $(\tau : \lceil P \rceil)$. Then τ is such that $\tau \leftrightarrow_{\mathsf{ty}} \{i = \tau_i\}^I$ where $(\tau_i : \lceil P_i \rceil)$ for all $i \in I$.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{i}}(\forall P.\tau, \{i = A_i\}^I) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{i}}(\{i = \forall P_i.\tau_i\}^I, \{i = A_i\}^I) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \{i = \mathsf{select}^{\mathsf{i}}(\forall P_i.\tau_i, A_i)\}^I \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \{i = \forall P'_i.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau_i, A^{\mathsf{s}}_i), A^{\mathsf{r}}_i)\}^I \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \forall P'.\{i = \mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau_i, A^{\mathsf{s}}_i), A^{\mathsf{r}}_i)\}^I \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \forall P'.\mathsf{select}^{\mathsf{i}}(\{i = \mathsf{select}^{\mathsf{b}}(\tau_i, A^{\mathsf{s}}_i)\}^I, A^{\mathsf{r}}) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \forall P'.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau, A^{\mathsf{s}}), A^{\mathsf{r}}) \end{array}$$

The first equation holds by definition of $\mathsf{select}^{\mathsf{i}}$, the second, fifth and sixth one by Lemma 7.23, the third one by induction hypothesis and the fourth one by definition of $\leftrightarrow_{\mathsf{ty}}$.

(6): By induction on the derivation of $(\mathsf{match}(P, A) = (P', A^{\mathsf{s}}, A^{\mathsf{r}}))$.

Case.

$$\frac{}{\mathsf{match}(*, \bar{A}) = (*, *, \bar{A})}$$

Suppose $(\tau : *)$. Then $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, *), *) \leftrightarrow_{\mathsf{ty}} \mathsf{expand}(\tau, *))$, by definition of $\mathsf{select}^{\mathsf{b}}$.

Case.

$$\frac{}{\mathsf{match}(\bar{P}, *) = (\bar{P}, *, \lceil\bar{P}\rceil)}$$

Suppose $(\tau : *)$. Then $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, \bar{P}), *) \leftrightarrow^{\perp}_{\mathsf{ty}} \mathsf{expand}(\tau, \bar{P}))$, by definition of $\mathsf{select}^{\mathsf{b}}$. Moreover, $\mathsf{expand}(\tau, \bar{P})$ is defined because $\bar{P}$ is an individual parameter.

Case.

$$\frac{\mathsf{match}(\bar{P}_j, \bar{A}) = (\bar{P}', \bar{A}^{\mathsf{s}}, A^{\mathsf{r}})}{\mathsf{match}(\mathsf{join}\{i = \bar{P}_i\}^I, (j, \bar{A})) = (\bar{P}', (j, \bar{A}^{\mathsf{s}}), A^{\mathsf{r}})} \quad \text{if } j \in I$$

Let $(P = \mathsf{join}\{i = \bar{P}_i\}^I)$. Suppose $(\tau : *)$.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, P), (j, \bar{A}^{\mathsf{s}})) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{b}}(\{i = \mathsf{expand}(\tau, \bar{P}_i)\}^I, (j, \bar{A}^{\mathsf{s}})) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, \bar{P}_j), \bar{A}^{\mathsf{s}}) \\ \leftrightarrow_{\mathsf{ty}} & \mathsf{expand}(\tau, \bar{P}') \end{array}$$

The first equation holds by definition of expand, the second one by definition of $\mathsf{select}^{\mathsf{b}}$, and the third one by induction hypotheses.

Case.

$$\frac{\mathsf{match}(P_i, A_i) = (P'_i, A^{\mathsf{s}}_i, A^{\mathsf{r}}_i) \quad \text{for all } i \in I}{\mathsf{match}(\{i = P_i\}^I, \{i = A_i\}^I) = (\{i = P'_i\}^I, \{i = A^{\mathsf{s}}_i\}^I, \{i = A^{\mathsf{r}}_i\}^I)}$$

Let $(P = \{i = P_i\}^I)$ and $(A^{\mathsf{s}} = \{i = A^{\mathsf{s}}_i\}^I)$. Suppose $(\tau : \lfloor P \rfloor)$. Then τ is such that $\tau \leftrightarrow_{\mathsf{ty}} \{i = \tau_i\}^I$ where $(\tau_i : \lfloor P_i \rfloor)$ for all $i \in I$.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, P), A^{\mathsf{s}}) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{select}^{\mathsf{b}}(\{i = \mathsf{expand}(\tau_i, P_i)\}^I, A^{\mathsf{s}}) \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \{i = \mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau_i, P_i), A^{\mathsf{s}}_i)\}^I \\ \leftrightarrow_{\mathsf{ty}} & \{i = \mathsf{expand}(\tau_i, P'_i)\}^I \\ \leftrightarrow^{\perp}_{\mathsf{ty}} & \mathsf{expand}(\tau, P') \end{array}$$

The first and fourth equation hold by Lemma 7.19, the second one by definition of $\mathsf{select}^{\mathsf{b}}$, and the third one by induction hypotheses. □

10.3 Reduction Rules for Terms

Let a binary relation $\mathcal{R} \subseteq \mathsf{Term} \times \mathsf{Term}$ be called *compatible* iff it satisfies the following rules:

$$\begin{array}{lcl} (M\ \mathcal{R}\ N) & \Rightarrow & ((\Lambda P.M)\ \mathcal{R}\ (\Lambda P.N)) \\ (M\ \mathcal{R}\ N) & \Rightarrow & ((M[A])\ \mathcal{R}\ (N[A])) \\ (M\ \mathcal{R}\ N) & \Rightarrow & ((\lambda x^{\tau}.M)\ \mathcal{R}\ (\lambda x^{\tau}.N)) \\ (M\ \mathcal{R}\ N) & \Rightarrow & ((M\ M')\ \mathcal{R}\ (N\ M')) \\ (M\ \mathcal{R}\ N) & \Rightarrow & ((M'\ M)\ \mathcal{R}\ (M'\ N)) \end{array}$$

Definition 10.5 (Term Reduction)**.** Let $\rightarrow_{\mathsf{b}}$ be the least compatible relation $\mathcal{R}$ that contains all instances of the rules (β), (Select), ($*_P$) and ($*_A$), below. Let $\rightarrow^*_{\mathsf{b}}$ be the reflexive and transitive closure of $\rightarrow_{\mathsf{b}}$.

$$\begin{array}{llcl}
(\beta) & ((\lambda x^{\tau}.M)N) & \mathcal{R} & (M[x := N]) \\
(\mathsf{Select}) & (\Lambda P.M)[A] & \mathcal{R} & \Lambda P'.((\mathsf{select}^{\mathsf{b}}(M, A^{\mathsf{s}}))[A^{\mathsf{r}}]), \\
 & \multicolumn{3}{l}{\text{if } \mathsf{match}(P, A) = (P', A^{\mathsf{s}}, A^{\mathsf{r}}) \text{ and neither } P \text{ nor } A \text{ is trivial}} \\
(*_P) & (\Lambda P.M) & \mathcal{R} & M, \quad \text{if } P \text{ is trivial} \\
(*_A) & (M[A]) & \mathcal{R} & M, \quad \text{if } A \text{ is trivial}
\end{array}$$

□

Example 10.6. Let the terms M and N and the type σ be as in Example 10.1. Consider this term:

$$M' \quad = \quad (\lambda y^{\sigma}.\, M)\, N$$

This term is well typed. Its type erasure (defined in Section 10.4) is $((\lambda y.\lambda x.\, y\, x)\,(\lambda x.x))$. The term M' reduces, by a β-reduction step, to the term

$$\Lambda P'.\lambda x^{\tau'}.\,(\Lambda\,\{j = P,\ l = P\}\,.\lambda x^{\{j=\tau,\ l=\tau\}}.x^{\{j=\tau,\ l=\tau\}})\ [A]\ x^{\tau'},$$

where P, P', τ, τ' and A are as in Example 10.1. □

Example 10.7. Consider the term $M = (\Lambda P_1.\Lambda P_2.\lambda x^{\tau}.x)[A]$, where

$$\begin{array}{ll}
P_1 & = \{m = \mathsf{join}\{j = *,\, k = *\},\, n = *\}, \\
P_2 & = \{m = *,\, n = \mathsf{join}\{h = *,\, l = *\}\}, \\
A & = \{m = *,\, n = (h, *)\}, \\
\tau & = \{m = \{j = \alpha_1,\, k = \alpha_2\},\, n = \{h = \beta_1,\, l = \beta_2\}\}.
\end{array}$$

The term M first rewrites to $\Lambda P_1.(\Lambda P_2.\lambda x^{\tau}.x)[A]$, by a (Select)-step, where P_1 and A pass through each other without interacting. By another (Select)-step, it rewrites to $\Lambda P_1.\Lambda P_2'.(\lambda x^{\sigma}.x)[A']$ where

$$\begin{array}{ll}
P_2' & = \{m = *,\, n = *\}, \\
A' & = \{m = *,\, n = *\}, \\
\sigma & = \{m = \{j = \alpha_1,\, k = \alpha_2\},\, n = \beta_1\}.
\end{array}$$

Finally, it rewrites to $\Lambda P_1.\lambda x^{\sigma}.x$, removing the trivial P_2' and A' by a ($*_P$)-step and a ($*_A$)-step. □

Theorem 10.8 (Subject Reduction)**.** *If* $(M \rightarrow_{\mathsf{b}} N)$ *and* $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa)$*, then* $(E \vdash^{\mathsf{B}} N : \tau \text{ at } \kappa)$*.* □

Proof. By induction on the proof tree of $(M \rightarrow_{\mathsf{b}} N)$. For ($\beta$), one uses Lemma 10.3. For ($*_A$) and ($*_P$), one uses Lemma 7.24. Here is the proof for (Select). Assume the following:

$$(E \vdash^{\mathsf{B}} (\Lambda P.M)[A] : \tau \text{ at } \kappa) \quad \text{and} \quad (\mathsf{match}(P, A) = (P', A^{\mathsf{s}}, A^{\mathsf{r}}))$$

From the first of these assumptions, by inversion of the typing rules, there is a τ' such that

$$(E \vdash^{\mathsf{B}} (\Lambda P.M) : \tau' \text{ at } \kappa) \quad \text{and} \quad (\mathsf{select}^{\mathsf{i}}(\tau', A) \leftrightarrow_{\mathsf{ty}} \tau)$$

From the first of these statements, by another inversion of the typing rules, there is a τ'' such that

$$(\mathsf{expand}(E, P) \vdash^{\mathsf{B}} M : \tau'' \text{ at } \lceil P \rceil) \quad \text{and} \quad (\tau' \leftrightarrow_{\mathsf{ty}} \forall P.\tau'') \quad \text{and} \quad (\kappa = \lfloor P \rfloor)$$

By Lemma 10.4 (4), it is the case that $(A^{\mathsf{s}} \lhd \lceil P \rceil)$. Therefore, by Lemma 8.6:

$$\begin{array}{l}\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(E, P), A^{\mathsf{s}}) \\ \quad \vdash^{\mathsf{B}} \mathsf{select}^{\mathsf{b}}(M, A^{\mathsf{s}}) : \mathsf{select}^{\mathsf{b}}(\tau'', A^{\mathsf{s}}) \text{ at } \mathsf{select}^{\mathsf{b}}(\lceil P \rceil, A^{\mathsf{s}})\end{array}$$

By Lemma 10.4 (2), it is the case that $(\mathsf{select}^{\mathsf{b}}(\lceil P \rceil, A^{\mathsf{s}}) = \lceil P' \rceil)$. Therefore,

$$\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(E, P), A^{\mathsf{s}}) \vdash^{\mathsf{B}} \mathsf{select}^{\mathsf{b}}(M, A^{\mathsf{s}}) : \mathsf{select}^{\mathsf{b}}(\tau'', A^{\mathsf{s}}) \text{ at } \lceil P' \rceil$$

By Lemma 10.4 (5), it is the case that $(\forall P'.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau'', A^{\mathsf{s}}), A^{\mathsf{r}}) \leftrightarrow^{\perp}_{\mathsf{ty}} \tau)$. In particular, $\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau'', A^{\mathsf{s}}), A^{\mathsf{r}})$ is defined. Let $\sigma = \mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau'', A^{\mathsf{s}}), A^{\mathsf{r}})$. Then, by $(\forall_{\mathsf{e}})$,

$$\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(E, P), A^{\mathsf{s}}) \vdash^{\mathsf{B}} (\mathsf{select}^{\mathsf{b}}(M, A^{\mathsf{s}}))[A^{\mathsf{r}}] : \sigma \text{ at } \lceil P' \rceil$$

By Lemma 10.4 (6), it is the case that $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(E, P), A^{\mathsf{s}}) \leftrightarrow_{\mathsf{ty}} \mathsf{expand}(E, P'))$. Therefore,

$$\mathsf{expand}(E, P') \vdash^{\mathsf{B}} (\mathsf{select}^{\mathsf{b}}(M, A^{\mathsf{s}}))[A^{\mathsf{r}}] : \sigma \text{ at } \lceil P' \rceil$$

Then, by $(\forall_{\mathsf{i}})$,

$$E \vdash^{\mathsf{B}} \Lambda P'.(\mathsf{select}^{\mathsf{b}}(M, A^{\mathsf{s}}))[A^{\mathsf{r}}] : \forall P'.\sigma \leftrightarrow_{\mathsf{ty}} \tau \text{ at } \lfloor P' \rfloor$$

By Lemma 10.4 (1), it is the case that $(\lfloor P' \rfloor = \kappa)$. Therefore,

$$E \vdash^{\mathsf{B}} \Lambda P'.(\mathsf{select}^{\mathsf{b}}(M, A^{\mathsf{s}}))[A^{\mathsf{r}}] : \tau \text{ at } \kappa.$$ □

10.4 Correspondence of Typed and Untyped Reduction

Substitution for untyped terms is defined as usual, and so is β-reduction:

$$(\beta) \qquad (\lambda x.M)N \quad \rightarrow_{\mathsf{i}} \quad M[x := N]$$

Let $\rightarrow^{*}_{\mathsf{i}}$ denote the reflexive and transitive closure of $\rightarrow_{\mathsf{i}}$.

Lemma 10.9.

1. *If* $\mathsf{expand}(M, P)$ *is defined, then* $(|\mathsf{expand}(M, P)| = |M|)$.
2. *If* $M[x :=^{\mathsf{b}} N]$ *is defined, then* $|M[x :=^{\mathsf{b}} N]| = |M[x :=^{\mathsf{i}} |N|]|$.
3. *If* $\mathsf{select}^{\mathsf{b}}(M, A)$ *is defined, then* $(|\mathsf{select}^{\mathsf{b}}(M, A)| = |M|)$. □

Proof. All statements, separately and in this order, by induction on the size of M. □

Theorem 10.10 (Soundness of Reduction)**.**
If $M, N \in \mathsf{Term}$ *and* $M \rightarrow_{\mathsf{b}} N$, *then* $|M| \rightarrow^{*}_{\mathsf{i}} |N|$. □

Proof. Obvious from the previous lemma. □

Lemma 10.11.

1. *Any sequence of* (Select), $(*_P)$ *and* $(*_A)$ *rewriting steps is terminating.*
2. *If* M *rewrites to* N *by* (Select), $(*_P)$ *or* $(*_A)$ *rewriting steps, then* $|M| = |N|$.
3. *If* $\mathsf{select}^{\mathsf{i}}(\forall P.\tau, A)$ *is defined, then so is* $\mathsf{match}(P, A)$.
4. *A well typed term that is free of* (Select), $(*_P)$ *or* $(*_A)$ *redexes is of the form*

   $$\Lambda P_1 \ldots \Lambda P_n.M[A_1] \ldots [A_m]$$

   where $n, m \geq 0$, *the* P_i*'s and* A_i*'s are not trivial, and* M *is either a variable, a* λ*-abstraction or an application.*

5. *If* $(\forall P.\tau' \leftrightarrow_{\mathsf{ty}} \tau \to \sigma)$*, then* P *is trivial.*

6. *If* $\mathsf{select}^{\mathsf{i}}(\tau \to \sigma, A)$ *is defined, then* A *is trivial.* □

Proof. To prove statement (1), define a weight function like this:

$$\|M\| \quad = \quad \left(\begin{array}{ll} & (\text{no. of occurrences of } \mathsf{join} \text{ in } M) \\ + & (\text{no. of subterms of the form of } \Lambda P.M' \text{ in } M) \\ + & (\text{no. of occurrences of } [\cdot] \text{ in } M) \end{array} \right)$$

An inspection of the rewriting rules shows that $\|M\| > \|N\|$, if M rewrites to N by one of (Select), $(*_P)$ and $(*_A)$. Statements (2) is obvious. Statement (3) is proved by induction on the structure of A, using the fact that P is an individual parameter if $\forall P.\tau$ is an individual type. Statement (4) follows from the well-typedness assumption and statement (3). Statement (5) is proved by induction on the structure of P, using that equivalent types have equal normal forms. Statement (6) is proved by induction on the structure of A. □

Theorem 10.12 (Completeness of Reduction)**.**
If M *is well typed and* $(|M| \to_{\mathsf{i}} \hat{N})$*, then there is a* λ^{B}*-term* N *such that* $(M \to_{\mathsf{b}}^* N)$ *and* $(|N| = \hat{N})$*.* □

Proof. The statement is proved by induction on the proof of $(|M| \to_{\mathsf{i}} \hat{N})$. The only interesting case is when $(|M| = (\lambda x.\hat{R})\hat{S})$ and $(\hat{N} = \hat{R}[x :=^{\mathsf{i}} \hat{S}])$. So suppose this is the case. We want to rewrite M to a term N such that $(|N| = \hat{R}[x :=^{\mathsf{i}} \hat{S}])$. To this end, first eliminate all (Select), $(*_P)$ and $(*_A)$ redexes from M. This terminates, by the previous lemma. The resulting term, let's call it M_1, is of the following form:

$$M_1 = \Lambda P_1 \ldots \Lambda P_n.(M_2\ M_3)[A_1] \ldots [A_m]$$

where $n, m \geq 0$, the P_i's and A_i's are not trivial, $(|M_2| = \lambda x.\hat{R})$ and $(|M_3| = \hat{S})$. The term M_2 is such that

$$M_2 = \Lambda P'_1 \ldots \Lambda P'_k.(\lambda x^{\sigma}.M_4)[A'_1] \ldots [A'_l]$$

where $k, l \geq 0$, the P'_i's and A'_i's are not trivial and $(|M_4| = \hat{R})$.

We now show that $(k = 0)$: The term M_2 must have a function type because $(M_2\ M_3)$ is well typed. Suppose, towards a contradiction, that $(k > 0)$. Then, M_2's type is equal to $(\forall P'_1.\tau)$ for some τ. Because P'_1 is not trivial, this contradicts the fact that M_2 has a function type, by Lemma 10.11 (5).

By a similar argument, which uses Lemma 10.11 (6), one shows that $l = 0$. Now, define N by

$$N = \Lambda P_1 \ldots \Lambda P_n.(M_4[x := M_3])[A_1] \ldots [A_m]$$

Then $(M \to_{\mathsf{b}}^* N)$ and $(|N| = \hat{R}[x :=^{\mathsf{i}} \hat{S}])$. □

References

[AB91] F. Alessi and F. Barbanera. Strong conjunction and intersection types. In A. Tarlecki, editor, *Proc. 16th Int'l Symp. Mathematical Foundations Computer Science (MFCS '91)*, volume 520 of *LNCS*, pages 64–73. Springer-Verlag, 1991.

[AT00] Torben Amtoft and Franklyn Turbak. Faithful translations between polyvariant flows and polymorphic types. In *Programming Languages & Systems, 9th European Symp. Programming*, volume 1782 of *LNCS*, pages 26–40. Springer-Verlag, 2000.

[Ban97] Anindya Banerjee. A modular, polyvariant, and type-based closure analysis. In ICFP '97 [ICFP97], pages 1–10.

[BM94] Franco Barbanera and Simone Martini. Proof-theoretical connectives and realizability. *Arch. Math. Logic*, 33:189–211, 1994.

[CDC80] Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Formal Logic*, 21(4):685–693, 1980.

[CLV01] Beatrice Capitani, Michele Loreti, and Betti Venneri. Hyperformulae, parallel deductions and intersection types. *Electronic Notes in Theoretical Computer Science*, 50, 2001. Proceedings of ICALP 2001 workshop: Bohm's Theorem: Applications to Computer Science Theory (BOTH 2001), Crete, Greece, 2001-07-13.

[DCGV97] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, and Betti Venneri. The "relevance" of intersection and union types. *Notre Dame J. Formal Logic*, 38(2):246–269, Spring 1997.

[DMTW97] Allyn Dimock, Robert Muller, Franklyn Turbak, and J. B. Wells. Strongly typed flow-directed representation transformations. In ICFP '97 [ICFP97], pages 11–24.

[DWM+01a] Allyn Dimock, Ian Westmacott, Robert Muller, Franklyn Turbak, and J. B. Wells. Functioning without closure: Type-safe customized function representations for Standard ML. In *Proc. 6th Int'l Conf. Functional Programming*, pages 14–25. ACM Press, 2001.

[DWM+01b] Allyn Dimock, Ian Westmacott, Robert Muller, Franklyn Turbak, J. B. Wells, and Jeffrey Considine. Program representation size in an intermediate language with intersection and union types. In *Types in Compilation, Third Int'l Workshop, TIC 2000*, volume 2071 of *LNCS*, pages 27–52. Springer-Verlag, 2001.

[DWM+01c] Allyn Dimock, Ian Westmacott, Robert Muller, Franklyn Turbak, J. B. Wells, and Jeffrey Considine. Program representation size in an intermediate language with intersection and union types. Technical Report BUCS-TR-2001-02, Comp. Sci. Dept., Boston Univ., March 2001. This is a version of [DWM+01b] extended with an appendix describing the CIL typed intermediate language.

[Gir72] J[ean]-Y[ves] Girard. *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur.* Thèse d'Etat, Université de Paris VII, 1972.

[Hin84] J. Roger Hindley. Coppo-Dezani types do not correspond to propositional logic. *Theoret. Comput. Sci.*, 28(1–2):235–236, January 1984.

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

[Jim95] Trevor Jim. What are principal typings and what are they good for? Tech. memo. MIT/LCS/TM-532, MIT, 1995.

[Kfo00] Assaf J. Kfoury. A linearization of the lambda-calculus. *J. Logic Comput.*, 10(3):411–436, 2000. Special issue on Type Theory and Term Rewriting. Kamareddine and Klop (editors).

[KMTW99] Assaf J. Kfoury, Harry G. Mairson, Franklyn A. Turbak, and J. B. Wells. Relating typability and expressibility in finite-rank intersection type systems. In *Proc. 1999 Int'l Conf. Functional Programming*, pages 90–101. ACM Press, 1999.

[KW94] A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus. In *Proc. 1994 ACM Conf. LISP Funct. Program.*, pages 196–207, 1994.

[KW99] Assaf J. Kfoury and 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.*, pages 161–174, 1999. Superseded by [KW04].

[KW03] Assaf J. Kfoury and J. B. Wells. Principality and type inference for intersection types using expansion variables. Supersedes [KW99], August 2003.

[KW04] Assaf J. Kfoury and J. B. Wells. Principality and type inference for intersection types using expansion variables. *Theoret. Comput. Sci.*, 311(1–3):1–70, 2004. Supersedes [KW99]. For omitted proofs, see the longer report [KW03].

[LE85] E. K. G. Lopez-Escobar. Proof-functional connectives. In C. Di Prisco, editor, *Methods of Mathematical Logic, Proceedings of the 6th Latin-American Symposium on Mathematical Logic, Caracas 1983*, volume 1130 of *Lecture Notes in Mathematics*, pages 208–221. Springer-Verlag, 1985.

[LRDR05] L. Liquori and Simona Ronchi Della Rocca. Towards an intersection typed system a la Church. In *Proc. 3rd Int'l Workshop Intersection Types & Related Systems (ITRS 2004)*, 2005. The ITRS '04 proceedings appears as vol. 136 (2005-07-19) of *Elec. Notes in Theoret. Comp. Sci.*

[Min89] G. E. Mints. The completeness of provable realizability. *Notre Dame J. Formal Logic*, 30(3):420–441, 1989.

[Pie91] Benjamin C. Pierce. Programming with intersection types, union types, and polymorphism. Technical Report CMU-CS-91-106, Carnegie Mellon University, February 1991.

[Pot80] Garrel Pottinger. A type assignment for the strongly normalizable λ-terms. In J. R[oger] Hindley and J[onathan] P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 561–577. Academic Press, 1980.

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

[RDRR01] Simona Ronchi Della Rocca and Luca Roversi. Intersection logic. In *Computer Science Logic, CSL '01*, pages 414–428. Springer-Verlag, 2001.

[Rey74] J. C. Reynolds. Towards a theory of type structure. In *Colloque sur la Programmation*, volume 19 of *LNCS*, pages 408–425. Springer-Verlag, 1974.

[Rey96] John C. Reynolds. Design of the programming language Forsythe. In P. O'Hearn and R. D. Tennent, editors, *Algol-like Languages*. Birkhauser, 1996.

[TDMW97] Franklyn Turbak, Allyn Dimock, Robert Muller, and J. B. Wells. Compiling with polymorphic and polyvariant flow types. In *Proc. First Int'l Workshop on Types in Compilation*, June 1997.

[Urz97] Paweł Urzyczyn. Type reconstruction in $\mathbf{F}_\omega$. *Math. Structures Comput. Sci.*, 7(4):329–358, 1997.

[vB95] Steffen J. van Bakel. Intersection type assignment systems. *Theoret. Comput. Sci.*, 151(2):385–435, 27 November 1995.

[Ven94] Betti Venneri. Intersection types as logical formulae. *J. Logic Comput.*, 4(2):109–124, April 1994.

[WDMT97] J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A typed intermediate language for flow-directed compilation. In *Proc. 7th Int'l Joint Conf. Theory & Practice of Software Development*, volume 1214 of *LNCS*, pages 757–771, 1997. Superseded by [WDMT02].

[WDMT02] J. B. Wells, Allyn Dimock, Robert Muller, and Franklyn Turbak. A calculus with polymorphic and polyvariant flow types. *J. Funct. Programming*, 12(3):183–227, May 2002. Supersedes [WDMT97].

[Wel94] J. B. Wells. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In *Proc. 9th Ann. IEEE Symp. Logic in Comput. Sci.*, pages 176–185, 1994. Superseded by [Wel99].

[Wel96] J. B. Wells. Typability is undecidable for F+eta. Tech. Rep. 96-022, Comp. Sci. Dept., Boston Univ., March 1996.

[Wel99] J. B. Wells. Typability and type checking in System F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1–3):111–156, 1999. Supersedes [Wel94].

[Wel02] J. B. Wells. The essence of principal typings. In *Proc. 29th Int'l Coll. Automata, Languages, and Programming*, volume 2380 of *LNCS*, pages 913–925. Springer-Verlag, 2002.