Branching Types*

J. B. Wells† Christian Haack

2002-08-18

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

$$(\lambda x.z(x(\lambda f u.fu))(x(\lambda v g.gv)))(\lambda y.yyy)$$

*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/.

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 $\sigma \,\&\, \tau$ 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 $(\textbf{for } \alpha \in \{\sigma, \tau\}.\lambda x{:}\alpha.\, \lambda y{:}\alpha.\, x)$ 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:

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

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 3, we will compare our approach to recent related work of Ronchi Della Rocca and Roversi [RDRR01] and Capitani, Loreti, and Venneri [CLV01].

2 Our Approach to the Problem: Our New System λ^{B}

This section presents our approach to solving the problem of the intersection introduction rule, our new system λ^{B} of *branching types*. To introduce λ^{B}, an 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 finally in λ^{B}.

2.1 Record-like Syntax and Pseudo-Grammars

Here are a few notational conventions that will be used throughout this article and which are necessary for following the examples presented 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 $\mathcal{I}$. We write a non-empty, finite function $\{\,(i, v_i) \,|\, i \in I\,\}$ as $\{i = v_i\}^I$. In other words, $\{i = v_i\}^I$ is 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. Pseudo-grammars are not proper grammars because they do not define proper syntax. Instead, they define sets that contain mathematical objects like finite functions.

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:

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

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 E range over the set Env of environments. The typing rules are shown in Figure 1. In this article, we refer to this system as λ^{I}.

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

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

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

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

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

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

This system is a fairly standard intersection type system where the intersection type constructor is n-ary, non-associative, non-commutative, and non-idempotent. Our record-like syntax for intersection types is slightly non-standard. We chose this syntax over a list-like syntax in order to simplify the technicalities when we relate our branching type system to this intersection type system (see Section 7). However, this choice is not essential for obtaining the correspondence.

Example 2.1. Consider the following untyped term:

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

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

σ are:

$$\begin{array}{rcl}
\tau & = & (\alpha \to \beta) \to \alpha \to \tau^z \to \beta \\
\tau^z & = & (\tau^w \to \beta) \to \beta \\
\tau^w & = & (\alpha \to \beta) \to \alpha \to \beta \\
\sigma & = & \gamma \to \wedge\{m = \gamma \to \gamma,\, n = \beta \to \beta\} \to \sigma^z \to \beta \\
\sigma^z & = & \wedge\{k = \sigma_k^w \to \beta,\, l = \sigma_l^w \to \beta\} \to \beta \\
\sigma_k^w & = & \gamma \to (\gamma \to \gamma) \to \beta \\
\sigma_l^w & = & \gamma \to (\beta \to \beta) \to \beta
\end{array}$$

□

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. In λ^{CIL}, a term constructor for *virtual records* is associated with the typing rule for intersection introduction. Virtual records differ from proper records in 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. 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. In this paper, we use the name λ^{CIL} to refer to the language given here, which is only a sublanguage of the real λ^{CIL}. The set $\mathsf{DupTerm}$ of *explicitly typed terms with duplicates* is defined by the following pseudo-grammar:

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

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:

$$\begin{array}{rclcrcl}
|x^\tau| & = & x & & |\lambda x^\tau.M| & = & \lambda x.|M| \\
|M\ N| & = & |M|\ |N| & & |\pi_i^\wedge\, M| & = & |M| \\
|\wedge\{i = M_i\}^I| & = & |M_j|, & \text{if } j \in I \text{ and } |M_j| = |M_i| \neq \bot \text{ for all } i \in I & & &
\end{array}$$

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

Example 2.2. Figure 3 shows the syntax tree of the λ^{CIL}-term that corresponds to the untyped term M with type $\wedge\{i = \tau,\, j = \sigma\}$ from Example 2.1. Note that the tree contains three duplicates of the subterm $(\lambda w.\ w\ x\ y)$ and two duplicates of M. □

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* [WDMT02].

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

In this paper, we propose the system λ^{B} of branching types — an intersection type system with explicit type annotations but without duplicates. The type annotations in λ^{B} contain the same information as the type annotations in λ^{CIL}. Thus, λ^{B} is suitable for the same kinds of uses as λ^{CIL}, such as customizing functions in multiple ways for different uses based

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

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

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

$$(\wedge_{\mathsf{i}}) \qquad \frac{E \vdash^{\mathsf{c}} M_i : \tau_i \text{ for all } i \in I}{E \vdash^{\mathsf{c}} \wedge\{i = M_i\}^I : \wedge\{i = \tau_i\}^I} \quad \text{if } |M_i| = |M_j| \neq \bot \text{ for all } i, j \in I$$

$$(\wedge_{\mathsf{e}}) \qquad \frac{E \vdash^{\mathsf{c}} M : \wedge\{i = \tau_i\}^I}{E \vdash^{\mathsf{c}} \pi^{\wedge}_j\ M : \tau_j} \quad \text{if } j \in I$$

Figure 2: λ^{CIL} — typing rules

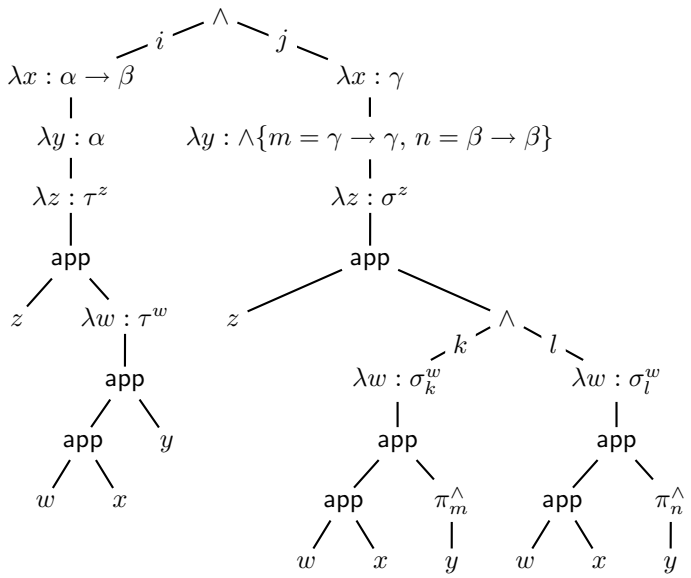


Figure 3: λ^{CIL} — a syntax tree

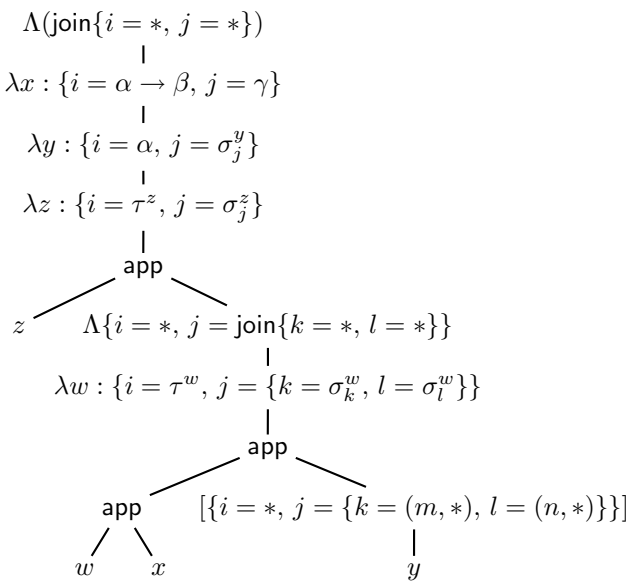


Figure 4: λ^{B} — a syntax tree

on type annotations. On the other hand, β-reduction is simpler in λ^{B} and there is no need for the use of parallel contexts in reduction.

The formal definition of λ^{B} is in Section 5. Here, the same example that was handled in Examples 2.1 and 2.2 for λ^{I} and λ^{CIL} is now used to convey the main features of λ^{B}.

EXAMPLE 2.3. In λ^{B}, the term M from Example 2.1 can be annotated to have the type ρ defined below, which corresponds to the type $\wedge\{i = \tau,\, j = \sigma\}$ from Example 2.1. The types $\tau, \tau^z, \tau^w, \sigma_k^w, \sigma_l^w$ are the same as in Example 2.1, but are repeated here for the reader's convenience.

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

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

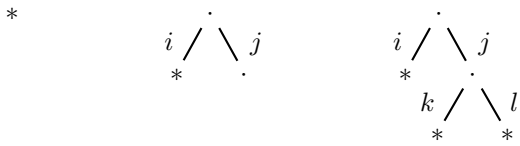
□

Example 2.3 will now be used throughout the rest of this section to illustrate the main features of λ^{B}.

Kinds (Branching Shapes). Every type and typing judgement of λ^{B} has a *kind*, which keeps track of its "branching shape" (cf. Sections 5.1.1 and 5.1.2). In λ^{B}, kinds 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 kinds 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 kinds that are used for typing judgements for the λ^{B} example are, from the root to the leaves, $*$, $\{i = *,\, j = *\}$, and $\{i = *,\, j = \{k = *,\, l = *\}\}$. These kinds can be viewed as the following three edge-labeled trees:



There is one additional kind that is relevant to the example. The binding type of y which is $\{i = \alpha,\, j = \sigma^y_j\}$ is equivalent (the equivalence is discussed further below) to a type of the shape $\forall P.\rho'$ where the kind of ρ' is $\{i = *,\, j = \{m = *, n = *\}\}$.

Branching Types. In the λ^{B} tree, all of the reasoning done in the i (left) branch of the λ^{CIL} tree has been placed inside the label i in the λ^{B} types and the reasoning done in the j branch of the λ^{CIL} tree has been placed inside the label j in the λ^{B} types. Similarly, the reasoning done in the k (left) and l (right) 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 is the *branching type* $\{i = \tau^w,\, j = \{k = \sigma^w_k,\, l = \sigma^w_l\}\}$ which has the kind $\{i = *,\, j = \{k = *,\, l = *\}\}$. 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. The typing rules allow replacing a derived result type by one that is equivalent to it by an equivalence relation named $\simeq$. This equivalence relation is obtained from a notion of reduction on types named $\succ$. (Details can be found in Definition 5.4.) 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:

$$\{i = \sigma_i\}^I \to \{i = \tau_i\}^I \succ \{i = \sigma_i \to \tau_i\}^I$$

To see why this rule is necessary, consider the subterm $\mathsf{app}(w, x)$ from 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 $\rho' \to \rho''$ where ρ' and ρ'' are some types. 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 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:

$$\begin{aligned} & \{i = \tau^w,\, j = \{k = \sigma^w_k,\, l = \sigma^w_l\}\} \\ = {} & \{i = \tau^w,\, j = \{k = \gamma \to (\gamma \to \gamma) \to \beta,\, l = \gamma \to (\beta \to \beta) \to \beta\}\} \\ \prec {} & \{i = \tau^w,\, j = \{k = \gamma, l = \gamma\} \to \{k = (\gamma \to \gamma) \to \beta,\, l = (\beta \to \beta) \to \beta\}\} \\ = {} & \{i = (\alpha \to \beta) \to \alpha \to \beta,\, j = \{k = \gamma, l = \gamma\} \to \{k = (\gamma \to \gamma) \to \beta,\, l = (\beta \to \beta) \to \beta\}\} \\ \prec {} & \quad \{i = \alpha \to \beta,\, j = \{k = \gamma, l = \gamma\}\} \\ & \to \{i = \alpha \to \beta,\, j = \{k = (\gamma \to \gamma) \to \beta,\, l = (\beta \to \beta) \to \beta\}\} \\ = {} & \rho^w_{\mathsf{d}} \to \rho^w_{\mathsf{c}} \end{aligned}$$

Hence, although w's type is not equal to a function type, it is equal to a function type modulo our notion of type equivalence.

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 reduction 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.

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 vparameter* P which is a pattern indicating what possible *type selection arguments* are valid to supply. Each parameter P has 2 kinds, its *inner kind* $\lfloor P \rfloor$ and its *outer kind* $\lceil P \rceil$. The kind of a type $\forall P.\rho'$ is $\lfloor P \rfloor$ and the kind 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 are not equivalent because there is a type selection abstraction intervening between the two occurrences of x. 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 5.2.1.) 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 reduction rules for type selection are the following:

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

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 $\{i = \alpha,\, j = \sigma_j^y\}$ and has kind $\{i = *,\, j = *\}$. Because the leaf occurrence of y must have kind $\{i = *,\, j = \{k = *,\, l = *\}\}$, the type at that location is expanded by the typing rules to be $\{i = \alpha,\, j = \{k = \sigma_j^y,\, l = \sigma_j^y\}\}$.

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:

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

The final type is the type actually used in figure 4. The type selection parameter of the final 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 reduction rules (see Definition 6.11) 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 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 reduction rules (see Definition 6.11) can simulate this single β-reduction step for untyped terms, although the simulation requires more than one reduction step. First, a sequence of reduction 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 reduction steps in detail. The term M first reduces to

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

by a (β_Λ)-step, where P_1 and A pass through each other without interacting. By another (β_Λ)-step, it reduces to $(\Lambda P_1.\Lambda P_2'.(\lambda x^{\tau'}.x)[A'])\,[B]\,y$ where

$$\begin{aligned} P_2' &= \{m = *,\, n = *\}, \quad \tau' = \{m = \{j = \alpha_1,\, k = \alpha_2\},\, n = \beta_1\} \\ A' &= \{m = *,\, n = *\}, \end{aligned}$$

Then, it reduces to $(\Lambda P_1.\lambda x^{\tau'}.x)\,[B]\,y$, removing the trivial P_2' and A' by a $(*_P)$-step and a $(*_A)$-step. 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 kind. Trivial parameters and arguments effectively do nothing, so it makes sense to simply remove them. By another (β_Λ)-step, followed by a $(*_P)$-step and a $(*_A)$-step, the term then reduces to $(\lambda x^{\tau''}.x)\,y$, where $\tau'' = \{m = \alpha_1,\, n = \beta_1\}$, matching exactly the type of y as one would expect. Finally, the term reduces to y by a (β)-step.

3 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. Much of the proof burden for the IL system (corresponding to a large portion of this paper) is inside the 3-word proof ("by structural induction") of their lemma 4 in the calculation of $S(\Pi, \Pi')$ where S is not constructively specified. 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.

4 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.

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

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.

4.1 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

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

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 $\{\, x \,|\, f(x) \text{ is defined} \,\}$ and is denoted by $\mathsf{dom}(f)$. The range of f is the set $\{\, f(x) \,|\, x \in \mathsf{dom}(f) \,\}$ and is denoted by $\mathsf{ran}(f)$. The expression $f[x \mapsto y]$ denotes the partial function $\{\, (x', y) \in f \,|\, x' \neq x \,\} \cup \{(x, y)\}$. 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. For all sets $X_1, \ldots, X_n$, the function smash is defined as follows:

$$\mathsf{smash} \quad : \quad X_1 \times \cdots \times X_n \to (X_1 \times \cdots \times X_n)_\bot$$

$$\mathsf{smash}(x_1, \ldots, x_n) \quad = \quad \begin{cases} \bot & \text{if } x_i = \bot \text{ for some } i \text{ in } \{1 \ldots n\} \\ (x_1, \ldots, x_n) & \text{otherwise} \end{cases}$$

In expressions that involve partial functions, we usually omit smashes and let the context resolve ambiguities. For example, if f is a partial function from $X \times X \times X$ to X, g is a partial function from X to X, and x, y, z are elements of X, then $f(g(x), g(y), z)$ is an abbreviation for $f(\mathsf{smash}(g(x), g(y), z))$.

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:

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

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.)

5 The Branching Type System λ^{B}

5.1 Types and Kinds

5.1.1 Kinds

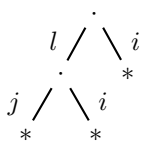
The set Kind of *kinds* is defined by the following pseudo-grammar:

$$\kappa \in \mathsf{Kind} \quad ::= \quad * \;\mid\; \{i = \kappa_i\}^I$$

One may view kinds as edge-labeled trees. For example, the kind

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

may be viewed as the following tree:



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

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

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

Lemma 5.1. *The relation $\leq$ is a partial order on the set of kinds.* □

5.1.2 Types

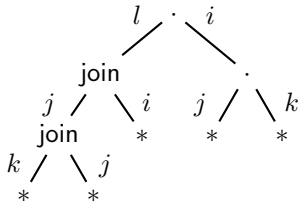
The sets Parameter of *type selection parameters* and IndParameter of *individual type selection parameters* are defined by the following pseudo-grammar:

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

Like kinds, 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:

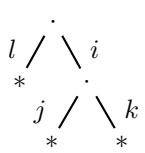


Note that if an internal node is labeled by join, then all the internal nodes below that node must also be labeled by join. Individual type selection parameters are those type selection parameters where *all* internal nodes are labeled by join.

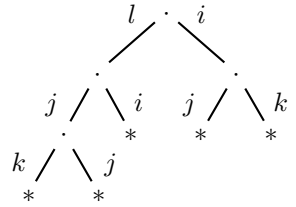
Given a parameter P, let P's *inner kind* $\lfloor P \rfloor$ and *outer kind* $\lceil P \rceil$ be defined as follows:

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

If P is the type selection parameter from above, then the following tree depicts its inner kind



and the following one its outer kind



Lemma 5.2. $\lfloor P \rfloor \leq \lceil P \rceil$. □

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

Note also that type variables are always of kind $*$. We considered the idea of allowing type variables of higher kinds. However, this would have complicated the system by requiring type environments (defined later in Section 5.4) to contain kinds of type variables in addition to types of term variables and to be sequences rather than finite maps. Furthermore, the effect of a higher-kind 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 kind 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 7. To get an idea of the relation, consider the following example:

Example 5.3. 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}:

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

5.1.3 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 *compatible* iff it satisfies the following rules:

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

Definition 5.4 (Type Equivalences). Let $\succ$ be the least compatible relation that contains all instances of the rules (1) through (3), below. Let $\succeq$ denote the reflexive and transitive closure of $\succ$, and $\simeq$ 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 $\succ$ is also a reduction relation, so the usual terminology of reduction (redex, contract, contractum, etc.) can be used with $\succ$. A type τ is called *irreducible* if there is no type σ such that $\tau \succ \sigma$. For any type τ, let $\mathsf{nf}(\tau)$ denote the unique irreducible type σ such that $\tau \succeq \sigma$. (Existence and uniqueness of $\mathsf{nf}(\tau)$ are proved in Lemma 5.9 below.) □

Lemma 5.5. *Let $\mathcal{R}$ range over $\{\succ, \succeq, \simeq\}$. If $(\tau\ \mathcal{R}\ \sigma)$, then $(\tau : \kappa)$ iff $(\sigma : \kappa)$.* □

Lemma 5.6. *If $(\{i = \tau_i\}^I \succeq \sigma)$, then σ is of the form $\{i = \sigma_i\}^I$ where $\tau_i \succeq \sigma_i$ for all $i \in I$.* □

Lemma 5.7 (Termination). *There is no infinite sequence $(\tau_n)_{n\in\mathbb{N}}$ such that $\tau_n \succ \tau_{n+1}$ 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 reduction rules shows that $\tau \succ \sigma$ implies $\|\tau\| > \|\sigma\|$. Therefore, every reduction sequence is finite. □

Lemma 5.8 (Confluence).

1. *If $\tau_1 \succ \tau_2$ and $\tau_1 \succ \tau_3$ where $\tau_2 \neq \tau_3$, then there is a type τ_4 such that $\tau_2 \succ \tau_4$ and $\tau_3 \succ \tau_4$.*
2. *If $\tau_1 \succeq \tau_2$ and $\tau_1 \succeq \tau_3$, then there is a type τ_4 such that $\tau_2 \succeq \tau_4$ and $\tau_3 \succeq \tau_4$.* □

Proof Sketch. An inspection of the reduction rules shows the following: Whenever a redex τ contains another redex σ, then σ still occurs in the contractum of τ. Moreover, σ doesn't get duplicated in the contraction step. For this reason, statement (1) holds. Statement (2) follows from (1) by the usual argument. Note that Lemma 5.7 is not needed because of the strength of (1). □

Lemma 5.9 (Unique Normal Forms).
For every type τ there is a unique irreducible type σ such that $\tau \succeq \sigma$. □

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

Lemma 5.10. *$(\tau \simeq \sigma)$ if and only if $(\mathsf{nf}(\tau) = \mathsf{nf}(\sigma))$.* □

Proof. This follows immediately from Lemma 5.9. □

5.1.4 A Grammar for Normal Types

An important property of irreducible types is that their top-level structure reflects the top-level structure of their kinds. 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 5.12 below, normality and irreducibility are equivalent for well formed types.

Lemma 5.11. *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 5.12.

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.* □

5.1.5 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 $(\sigma \simeq \tau \to \tau')$. Lemma 5.14(2) gives a simple decision algorithm for this question. Lemma 5.13(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 $\simeq$, although we do not do so because we do not need such an algorithm in this paper.

Lemma 5.13.

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

Proof. Statement (1) follows from Lemmas 5.10 and 5.6. Statement (2) is proved by induction on the size of $(\tau \to \tau')$'s $\succ$-reduction graph. Statement (3) is proved by induction on the size of $(\forall P.\tau)$'s $\succ$-reduction graph. □

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

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

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 5.14. *Suppose σ is well formed.*

1. *If $(\sigma \simeq \{i = \tau_i\}^I)$, 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 $(\sigma \simeq \mathsf{tdom}(\sigma) \to \mathsf{tcodom}(\sigma))$.
   (c) If $(\sigma \simeq \tau \to \tau')$, then $\mathsf{tdom}(\sigma)$ is defined.
3. *(a) If $\mathsf{split}(P, \sigma)$ is defined, then $(\sigma \simeq \forall P.\, \mathsf{split}(P, \sigma))$.*
   (b) If $(\sigma \simeq \forall P.\tau)$, then $\mathsf{split}(P, \sigma)$ is defined. □

Proof. (1): Follows from Lemmas 5.10 and 5.6.
(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. Applying the type equivalence rules to α any number of times and in any direction yields only terms 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. Applying the type equivalence rules to $(\forall \bar{P}^\bullet.\mu)$ any number of times and in any direction yields only terms of the form $(\forall * \ldots \forall * .\forall \bar{P}^\bullet.\sigma')$.
Case, $(\sigma = \{i = \mu_i\}^I)$**:** Suppose $(\sigma \simeq \tau \to \tau')$. Then $(\sigma = \mathsf{nf}(\tau \to \tau'))$, by Lemma 5.10. 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 $(\mu_i \simeq \tau_i \to \tau_i')$ 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 $(\sigma \simeq \forall \bar{P}^\bullet.\tau)$. Applying the type equivalence rules to $(\forall \bar{P}^\bullet.\tau)$ any number of times and in any direction yields only terms of the form $(\forall * \ldots \forall * .\forall \bar{P}^\bullet.\tau')$. Therefore and because σ is assumed to be normal, it must be of the form $(\sigma = \forall \bar{P}^\bullet.\tau')$. Then $(\mathsf{split}(\bar{P}^\bullet, \sigma) = \tau')$.
Case, $(P = \{i = P_i\}^I)$**:** Suppose σ is well formed and $(\sigma \simeq \forall P.\tau)$. Then $\forall P.\tau$ is well formed, too, by Lemma 5.5. Moreover, τ is normal, by assumption, and $(\tau : \lceil P \rceil)$. Therefore, τ is of

the form $\{i = \tau_i\}^I$. By Lemma 5.5, it is the case that $(\sigma : \lfloor P \rfloor)$. Then, because σ is assumed normal, it is the case that σ is of the form $\{i = \mu_i\}^I$. Because $(\sigma \simeq \forall P.\tau \simeq \{i = \forall P_i.\tau_i\}^I)$, it is the case that $(\mu_i \simeq \forall P_i.\tau_i)$ for all $i \in I$, by Lemma 5.13(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. □

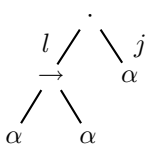
5.2 Expansion and Selection for Types

5.2.1 Type Expansion

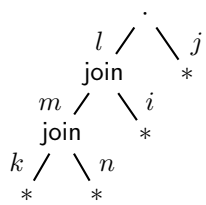
For the typing rules, we need to define some auxiliary operations on types. Let the partial function expand from $\mathsf{Ty} \times \mathsf{Parameter}$ to Ty be inductively defined by the equations below. Applying expand to the pair (τ, P) adjusts the type τ of branching shape $\lfloor P \rfloor$ to 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.

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

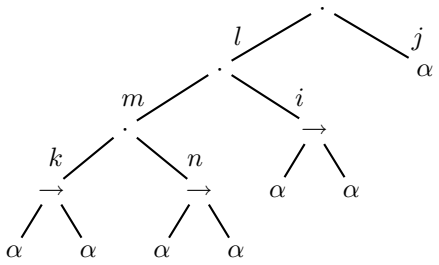
Example 5.15. Suppose $\tau = \{l = \alpha \to \alpha,\ j = \alpha\}$. This type is depicted by the following tree:



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

Lemma 5.17.

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 5.11. Then, for the case where τ is not normal, one uses the fact that normalization preserves kinds (Lemma 5.5). The proof of part (2) follows the same strategy. □

Lemma 5.18. *If* $(\mathsf{expand}(\tau, P) \simeq \mathsf{expand}(\tau', P))$*, then* $(\tau \simeq \tau')$*.* □

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 5.19.

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

Proof. Parts (1) and (2) are obvious for normal types. For non-normal types, they follow from Lemma 5.6. 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 $\succ$-reduction-graph of $(\sigma \to \tau)$, using parts (1) and (2). □

5.2.2 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:

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

We define two relations that assign kinds to arguments:

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

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

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 5.20. *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, inductively by the equations below. 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 5.21. *If $f \in \{\mathsf{select}^{\mathsf{i}}, \mathsf{select}^{\mathsf{b}}\}$ and $f(\tau, A) = \tau'$, then τ' is normal.* □

Lemma 5.22.

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 5.23. *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) *There is a family $(A'_i)_{i \in I}$ of type selection arguments such that $(A = \{i = A'_i\}^I)$.*
   (b) *There is a label j in I and an individual type selection argument $\bar{A}$ such that $A = (j, \bar{A})$.*
3. *$f(\{i = \tau_i\}^I, \{i = A_i\}^I) \simeq^{\perp} \{i = f(\tau_i, A_i)\}^I$.*
4. *If $(j \in I)$ and $(P = \mathsf{join}\{i = \bar{P}_i\}^I)$, then $(\mathsf{select}^{\mathsf{i}}(P, (j, \bar{A})) \simeq^{\perp} \mathsf{select}^{\mathsf{i}}(\forall \bar{P}_j.\tau_j, \bar{A}))$.*
5. *If $(j \in I)$, then $(\mathsf{select}^{\mathsf{b}}(\{i = \tau_i\}^I, (j, \bar{A})) \simeq^{\perp} \mathsf{select}^{\mathsf{b}}(\tau_j, \bar{A}))$.*
6. *$(f(\sigma \to \tau, A) \simeq^{\perp} f(\sigma, A) \to f(\tau, A))$.* □

Proof. The proof follows the same strategy as the proof of Lemma 5.19. □

5.3 Trivial Parameters and Arguments

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

Lemma 5.24 (Trivial Parameters and Arguments).

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

Proof. Statements (1), (3) and (4) are proved by induction on the structure of P, and statement (2) by induction on the structure of A. □

5.4 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:

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

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

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 judgements are of the form:

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

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

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

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

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

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

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

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

Figure 5: λ^{B} — typing rules

Lemma 5.25.

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* $(E \simeq E')$, *then* $(E' \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa)$. □

Proof. Statement (1) is proved by induction on the derivation of $(E \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa)$, using the fact that $\mathsf{select}^{\mathsf{i}}$ preserves kinds (Lemma 5.22), and that expand reflects kinds in the way expressed in Lemma 5.17. Statement (2) is proved by induction on the structure of $(E \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa)$, using the fact that $\simeq$ preserves kinds (Lemma 5.5). □

Proposition 5.26 (Unicity of Typing). *If* $(E \vdash^{\mathsf{B}} M : \tau \ \mathsf{at}\ \kappa)$ *and* $(E' \vdash^{\mathsf{B}} M : \tau' \ \mathsf{at}\ \kappa')$, *then* $(\tau \simeq \tau')$, $(\kappa = \kappa')$ *and* $(E(x) \simeq E'(x))$ *for all free variables* x *of* M. □

Proof. By induction on the structure of M, using the fact that expand reflects type equivalence (Lemma 5.18). □

EXAMPLE 5.27. 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 6.3) 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 judgement

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

is derivable in our system λ^{B}. What follows is the derivation of this judgement, presented in a goal-directed style. Note that in the reduction 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, the reduction 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.	$\vdash^{\mathsf{B}}\ M\ :\ (\sigma \to \sigma'')\ \mathsf{at}\ *$	By $(\to_{\mathsf{i}})$
2.	$y : \sigma\ \vdash^{\mathsf{B}}\ (\Lambda P'.\lambda x^{\tau'}.\, y^{\sigma'}\ [A]\ x^{\tau'})\ :\ \sigma''\ \ \mathsf{at}\ \ *$	By $(\forall_{\mathsf{i}})$
3.	$y : \sigma'\ \vdash^{\mathsf{B}}\ (\lambda x^{\tau'}.\, y^{\sigma'}\ [A]\ x^{\tau'})\ :$ $\{j = \alpha \to \alpha,\, l = \beta \to \beta\}\ \ \mathsf{at}\ \ \{j = *,\, l = *\}$	By $(\simeq)$
4.	$y : \sigma'\ \vdash^{\mathsf{B}}\ (\lambda x^{\tau'}.\, y^{\sigma'}\ [A]\ x^{\tau'})\ :\ (\tau' \to \tau')\ \ \mathsf{at}\ \ \{j = *,\, l = *\}$	By $(\to_{\mathsf{i}})$
5.	$y : \sigma',\, x : \tau'\ \vdash^{\mathsf{B}}\ (y^{\sigma'}\ [A]\ x^{\tau'})\ :\ \tau'\ \ \mathsf{at}\ \ \{j = *,\, l = *\}$	By $(\to_{\mathsf{e}})$
6.r.	$y : \sigma',\, x : \tau'\ \vdash^{\mathsf{B}}\ x^{\tau'}\ :\ \tau'\ \ \mathsf{at}\ \ \{j = *,\, l = *\}$	By (ax)
6.l.	$y : \sigma',\, x : \tau'\ \vdash^{\mathsf{B}}\ (y^{\sigma'}\ [A])\ :\ (\tau' \to \tau')\ \ \mathsf{at}\ \ \{j = *,\, l = *\}$	By $(\forall_{\mathsf{e}})$
6.l.1	$y : \sigma',\, x : \tau'\ \vdash^{\mathsf{B}}\ y^{\sigma'}\ :\ \sigma'\ \ \mathsf{at}\ \ \{j = *,\ l = *\}$	By (ax)

□

6 Term Reduction in λ^{B}

6.1 Auxiliary Operations Used in the Reduction Rules

In this section, we define auxiliary operations that are needed for the statements of the term reduction rules.

6.1.1 Term Expansion

In order to 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 $\mathsf{Parameter} \cup \mathsf{Argument} \cup \mathsf{Kind}$:

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

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 6.1. 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 6.3) 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 6.2 (Properties of expand).

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

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 $(\mathsf{expand}(\mathsf{expand}(\tau, P'), P) \simeq \mathsf{expand}(\mathsf{expand}(\tau, P), \mathsf{expand}(P', P)))$.

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,

$$\mathsf{expand}(\mathsf{expand}(\tau, P'), *) \ \simeq^{\perp} \ \mathsf{expand}(\tau, P') \ \simeq^{\perp} \ \mathsf{expand}(\mathsf{expand}(\tau, *), \mathsf{expand}(P', *))$$

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.

$$\begin{array}{rl}
 & \mathsf{expand}(\mathsf{expand}(\tau, P'), P) \\
\simeq^{\perp} & \{i = \mathsf{expand}(\mathsf{expand}(\tau, P'), \bar{P}_i)\}^I \\
\simeq & \{i = \mathsf{expand}(\mathsf{expand}(\tau, \bar{P}_i), \mathsf{expand}(P', \bar{P}_i))\}^I \\
\simeq^{\perp} & \mathsf{expand}(\{i = \mathsf{expand}(\tau, \bar{P}_i)\}^I, \{i = \mathsf{expand}(P', \bar{P}_i)\}^I) \\
\simeq^{\perp} & \mathsf{expand}(\mathsf{expand}(\tau, P), \mathsf{expand}(P', P))
\end{array}$$

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 $i \in I$.

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

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 $(\mathsf{expand}(\forall P'.\tau, P) \simeq \forall(\mathsf{expand}(P', P)).(\mathsf{expand}(\tau, P)))$.

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.

$$\mathsf{expand}(\forall P'.\tau, *) \ \simeq \ \forall P'.\tau \ \simeq \ \forall(\mathsf{expand}(P', *)).(\mathsf{expand}(\tau, *))$$

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

$$\mathsf{expand}(\forall *.\tau, P) \ \simeq^\perp \ \mathsf{expand}(\tau, P) \ \simeq^\perp \ \forall(\mathsf{expand}(*, P)).(\mathsf{expand}(\tau, P))$$

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.

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

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 $\simeq$.
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 $i \in I$.

$$\begin{array}{ll} & \mathsf{expand}(\forall P'.\tau, P) \\ \simeq^\perp & \mathsf{expand}(\{i = \forall P'_i.\tau_i\}^I, P) \\ \simeq^\perp & \{i = \mathsf{expand}(\forall P'_i.\tau_i, P_i)\}^I \\ \simeq & \{i = \forall(\mathsf{expand}(P'_i, P_i)).(\mathsf{expand}(\tau_i, P_i))\}^I \\ \simeq^\perp & \forall\{i = \mathsf{expand}(P', P_i)\}^I.\{i = \mathsf{expand}(\tau_i, P_i)\}^I \\ \simeq^\perp & \forall(\mathsf{expand}(P', P)).(\mathsf{expand}(\tau, P)) \end{array}$$

The first and last of these equations hold by definition of expand, the second one by Lemma 5.19, the third one by induction hypotheses and the fourth one by definition of $\simeq$.
(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 $(\mathsf{expand}(\tau', P) \simeq \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)))$.

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.

$$\mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, *), \mathsf{expand}(A, *)) \simeq^{\perp} \mathsf{select}^{\mathsf{i}}(\tau, A) = \tau' \simeq \mathsf{expand}(\tau', *)$$

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.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)) \\ \simeq^{\perp} & \mathsf{select}^{\mathsf{i}}(\{i = \mathsf{expand}(\tau, \bar{P}_i)\}^I, \{i = \mathsf{expand}(A, \bar{P}_i)\}^I) \\ \simeq^{\perp} & \{i = \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, \bar{P}_i), \mathsf{expand}(A, \bar{P}_i))\}^I \\ \simeq & \{i = \mathsf{expand}(\tau', \bar{P}_i)\}^I \\ \simeq^{\perp} & \mathsf{expand}(\tau', P) \end{array}$$

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 $i \in I$.

$$\begin{array}{ll} & \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)) \\ \simeq^{\perp} & \mathsf{select}^{\mathsf{i}}(\{i = \mathsf{expand}(\tau_i, P_i)\}^I, \{i = \mathsf{expand}(A_i, P_i)\}^I) \\ \simeq^{\perp} & \{i = \mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau_i, P_i), \mathsf{expand}(A_i, P_i))\}^I \\ \simeq & \{i = \mathsf{expand}(\tau'_i, P_i)\}^I \\ \simeq^{\perp} & \mathsf{expand}(\tau', P) \end{array}$$

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

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 6.2, (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
$\mathsf{expand}(E, P)$

$\vdash^{\mathsf{B}} \mathsf{expand}(\Lambda P'.M, P) : \mathsf{expand}(\forall P'.\tau, P)$ at $\mathsf{expand}(\lfloor P' \rfloor, P)$
By Lemma 6.2, (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
$\mathsf{expand}(E, P) \vdash^{\mathsf{B}} \mathsf{expand}(M, P) : \mathsf{expand}(\tau, P)$ at $\mathsf{expand}(\kappa, P)$ By ind. hyp.
$\mathsf{select}^{\mathsf{i}}(\mathsf{expand}(\tau, P), \mathsf{expand}(A, P)) \simeq \mathsf{expand}(\tau', P)$ By Lemma 6.2 (10)
$\mathsf{expand}(E, P) \vdash^{\mathsf{B}} \mathsf{expand}(M[A], P) : \mathsf{expand}(\tau', P)$ at $\mathsf{expand}(\kappa, P)$
By $(\forall_{\mathsf{e}})$ and definition of expand for terms
□

Corollary 6.4. *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 6.3 and Lemma 6.2 (7). □

6.1.2 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:

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

Let the *application of a substitution* s to a term M be defined inductively by the following equations.

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

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

Example 6.5. Let N, P, P', τ and σ be as in Example 6.1. 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 6.3) is $(\lambda x.\, y\, x)$. Substituting N for y in M results in

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

□

Let typing judgements for substitutions be defined as follows:

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

Lemma 6.6. *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 6.4. □

Lemma 6.7. *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 6.6. □

6.1.3 Selection for Terms

The preceding treatment of substitution prepares for the definition of β-reduction of terms of the form $((\lambda x^{\tau}.M)N)$. We also need to define reduction of terms of the form $(\Lambda P.M)[A]$. To this end, we extend $\mathsf{select}^{\mathsf{b}}$ to parameters, arguments and kinds, inductively by the following equations, where X ranges over $\mathsf{Parameter} \cup \mathsf{Argument} \cup \mathsf{Kind}$:

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

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

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

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

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

Lemma 6.8 (Properties of $\mathsf{select}^{\mathsf{b}}$).

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

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 6.2. □

Lemma 6.9. *If* $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa)$ *and* $(A \triangleleft \kappa)$,
then $(\mathsf{select}^{\mathsf{b}}(E, A) \vdash^{\mathsf{B}} \mathsf{select}^{\mathsf{b}}(M, A) : \mathsf{select}^{\mathsf{b}}(\tau, A) \text{ at } \mathsf{select}^{\mathsf{b}}(\kappa, A))$. □

Proof. By induction on the derivation of $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa)$, using Lemma 6.8. □

6.1.4 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 reduction rule (β_Λ) for type selection. The (β_Λ)-reduction 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.

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

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

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

Lemma 6.10 (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* $(\mathsf{select}^{\mathsf{i}}(\forall P.\tau, A) \ \simeq^{\perp} \ \forall P'.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau, A^{\mathsf{s}}), A^{\mathsf{r}}))$.
6. *If* $(\tau : \lfloor P \rfloor)$, *then* $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, P), A^{\mathsf{s}}) \ \simeq \ \mathsf{expand}(\tau, P'))$. □

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 $(\tau : *)$.

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

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 $\simeq$.
Case.

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

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

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

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 5.24, 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 $\tau \simeq \{i = \tau_i\}^I$ where $(\tau_i : \lceil \bar{P}_i \rceil)$ for all $i \in I$.

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

The first and third equation hold by Lemma 5.23, and the second one by induction hypotheses.
Case.

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

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

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

The first equation holds by definition of $\mathsf{select}^{\mathsf{i}}$, the second, fifth and sixth one by Lemma 5.23, the third one by induction hypothesis and the fourth one by definition of $\simeq$.
(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 $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, *), *) \simeq \mathsf{expand}(\tau, *))$, by definition of $\mathsf{select}^{\mathsf{b}}$.
Case.

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

Suppose $(\tau : *)$. Then $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, \bar{P}), *) \simeq^{\perp} \mathsf{expand}(\tau, \bar{P}))$, 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 $(\tau : *)$.

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

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 $\tau \simeq \{i = \tau_i\}^I$ where $(\tau_i : \lfloor P_i \rfloor)$ for all $i \in I$.

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

The first and fourth equation hold by Lemma 5.19, the second one by definition of $\mathsf{select}^{\mathsf{b}}$, and the third one by induction hypotheses. □

6.2 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:

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

Definition 6.11 (Term Reduction). Let $\rightarrow$ be the least compatible relation $\mathcal{R}$ that contains all instances of the rules (β_λ), (β_Λ), $(*_P)$ and $(*_A)$, below.

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

□

Example 6.12. Let the terms M and N and the type σ be as in Example 6.5. Consider this term:

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

This term is well typed. Its type erasure (defined in Section 6.3) is $((\lambda y.\lambda x.\, y\, x)\, (\lambda x.x))$. The term M' reduces, by a β_λ-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 6.5. □

EXAMPLE 6.13. Consider the term $M = (\Lambda P_1.\Lambda P_2.\lambda x^{\tau}.x)[A]$, where

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

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

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

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

Theorem 6.14 (Subject Reduction). *If* $(M \to N)$ *and* $(E \vdash^{\mathsf{B}} M : \tau \;\mathsf{at}\; \kappa)$, *then* $(E \vdash^{\mathsf{B}} N : \tau \;\mathsf{at}\; \kappa)$. □

Proof. By induction on the proof tree of $(M \to N)$. For (β_λ), one uses Lemma 6.7. For $(*_A)$ and $(*_P)$, one uses Lemma 5.24. Here is the proof for (β_Λ). Assume the following:

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

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

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

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

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

By Lemma 6.10 (4), it is the case that $(A^{\mathsf{s}} \triangleleft \lceil P \rceil)$. Therefore, by Lemma 6.9:

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

By Lemma 6.10 (2), it is the case that $(\mathsf{select}^{\mathsf{b}}(\lceil P \rceil, A^{\mathsf{s}}) = \lceil P' \rceil)$. Therefore,

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

By Lemma 6.10 (5), it is the case that $(\forall P'.\mathsf{select}^{\mathsf{i}}(\mathsf{select}^{\mathsf{b}}(\tau'', A^{\mathsf{s}}), A^{\mathsf{r}}) \simeq^{\perp} \tau)$. 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 $(\forall_{\mathsf{e}})$,

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

By Lemma 6.10 (6), it is the case that $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(E, P), A^{\mathsf{s}}) \simeq \mathsf{expand}(E, P'))$. Therefore,

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

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

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

By Lemma 6.10 (1), it is the case that $(\lfloor P' \rfloor = \kappa)$. Therefore,

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

□

6.3 Correspondence of Typed and Untyped Reduction

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

$$(\beta) \qquad (\lambda x.M)N \quad \to \quad M[x := N]$$

Let $\to^*$ denote the reflexive and transitive closure of $\to$. We define a map $|\cdot|$ from Term to $\mathsf{UntypedTerm}$ that erases type-annotations:

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

Lemma 6.15.

1. *If* $\mathsf{expand}(M, P)$ *is defined, then* $(|\mathsf{expand}(M, P)| = |M|)$.
2. *If* $M[x := N]$ *is defined, then* $|M[x := N]| = |M|[x := |N|]$.
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 6.16 (Soundness of Reduction).
If $M, N \in \mathsf{Term}$ *and* $M \to N$*, then* $|M| \to^* |N|$. □

Proof. Obvious from the previous lemma. □

Lemma 6.17.

1. *Any sequence of* (β_Λ), $(*_P)$ *and* $(*_A)$ *reduction steps is terminating.*
2. *If* M *reduces to* N *by* (β_Λ), $(*_P)$ *or* $(*_A)$ *reduction, 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* (β_Λ), $(*_P)$ *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* $(\forall P.\tau' \simeq \tau \to \sigma)$*, 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:

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

An inspection of the reduction rules shows that $\|M\| > \|N\|$, if M reduces to N by one of (β_Λ), $(*_P)$ 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 6.18 (Completeness of Reduction).
If M *is well typed and* $(|M| \to N')$*,*
then there is a λ^{B}*-term* N *such that* $(M \to^* N)$ *and* $(|N| = N')$. □

Proof. The statement is proved by induction on the proof of $(|M| \to N')$. The only interesting case is when $(|M| = (\lambda x.R)S)$ and $(N' = R[x := S])$. So suppose this is the case. We want to reduce M to a term N such that $(|N| = R[x := S])$. To this end, first eliminate all (β_Λ), $(*_P)$ 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, $(|M_2| = \lambda x.R)$ and $(|M_3| = S)$. 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 $(|M_4| = R)$.

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 6.17 (5).

By a similar argument, which uses Lemma 6.17 (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 $(M \to^* N)$ and $(|N| = R[x := S])$. □

7 Relating Branching Types to Intersection Types

7.1 From Branching Types to Intersection Types

Individual λ^{B}-types are closely related to 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 inductively by the following equations:

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

The function is extended to environments by $|E|(x) = |E(x)|$.

Theorem 7.1 (Soundness of $|\cdot|$, Special Version). *If* $(E \vdash^{\mathsf{B}} M : \tau\ \mathsf{at}\ *)$, *then* $(|E| \vdash^{\mathsf{i}} |M| : |\tau|)$. □

In the remainder of this section, we will prove this theorem. Note that the theorem only makes a claim about *individual* λ^{B}-typings. The theorem is a corollary of a more general theorem that applies to arbitrary λ^{B}-typings. To formulate this more general soundness theorem, we need to make some definitions.

Definition 7.2 (Maximal Paths of Kinds). An individual argument $\bar A$ is called a *maximal path* of a kind κ, if $(\mathsf{select}^{\mathsf{b}}(\kappa, \bar A) = *)$. We write $(\bar A \trianglelefteq \kappa)$ iff $\bar A$ is a maximal path of κ. $\mathsf{Maxpath}(\kappa)$ denotes the set of all maximal paths of κ. □

Lemma 7.3.

1. *If* $(\bar A \trianglelefteq \kappa)$, *then* $(\bar A \triangleleft \kappa)$.
2. *If* $(\bar A \trianglelefteq \lceil \bar P \rceil)$, *then* $(\mathsf{select}^{\mathsf{b}}(\mathsf{expand}(\tau, \bar P), \bar A) \simeq \tau)$. □

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 7.4 (Soundness of $|\cdot|$, General Version).
If $(E \vdash^{\mathsf{B}} M : \tau\ \mathsf{at}\ \kappa)$ *and* $(\bar{A} \trianglelefteq \kappa)$, *then* $(|\mathsf{select}^{\mathsf{b}}(E, \bar{A})| \vdash^{\mathsf{i}} |M| : |\mathsf{select}^{\mathsf{b}}(\tau, \bar{A})|)$. □

Proof of Theorem 7.1. Follows from Theorem 7.4, because $(* \trianglelefteq *)$, $(\mathsf{select}^{\mathsf{b}}(E, *) \simeq E)$ and $(\mathsf{select}^{\mathsf{b}}(\tau, *) \simeq \tau)$. □

The remainder of this section is devoted to the proof of Theorem 7.4. The proof is by lexicographical induction on the pair $\langle \mathsf{size}(M), \mathsf{size}(\kappa) \rangle$, where size is defined as follows:

$$\mathsf{size} \ : \ \mathsf{Term} \to \mathbb{N}$$

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

$$\mathsf{size} \ : \ \mathsf{Kind} \to \mathbb{N}$$

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

Lemma 7.5. *If* $(\mathsf{select}^{\mathsf{b}}(M, A) = N)$, *then* $(\mathsf{size}(M) = \mathsf{size}(N))$. □

Lemma 7.6. $\mathsf{select}^{\mathsf{b}}(\tau, (i, \bar{A})) \ \simeq^{\perp} \ \mathsf{select}^{\mathsf{b}}(\mathsf{select}^{\mathsf{b}}(\tau, (i, *)), \bar{A})$. □

Part (2) of the following lemma simulates the $(\forall_{\mathsf{e}})$-rule on λ^{I}-typing-judgements. Part (1) simulates an instance of the $(\forall_{\mathsf{i}})$-rule. (The simulation of this instance suffices for the proof of Theorem 7.4.)

Lemma 7.7.

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

Proof. (1): It suffices to prove the statement for normal τ. So suppose that this is the case. The proof is by induction on the structure of $\bar{P}$.
Case, $\bar{P} = *$**:** Suppose $(\tau : *)$ and $(E \vdash^{\mathsf{i}} M : |\mathsf{select}^{\mathsf{b}}(\tau, \bar{A})|)$ for all $\bar{A} \in \mathsf{Maxpath}(*)$. Because $(* \in \mathsf{Maxpath}(*))$, it is then the case that $(E \vdash^{\mathsf{i}} M : |\mathsf{select}^{\mathsf{b}}(\tau, *)|)$. But $(\mathsf{select}^{\mathsf{b}}(\tau, *) \simeq \forall *.\tau)$.
Case, $\bar{P} = \mathsf{join}\{i = \bar{P}_i\}^I$**:** Suppose $(\tau : \lceil \bar{P} \rceil)$ and $(E \vdash^{\mathsf{i}} M : |\mathsf{select}^{\mathsf{b}}(\tau, \bar{A})|)$ for all $\bar{A} \in \mathsf{Maxpath}(\lceil \bar{P} \rceil)$. Because we assumed τ normal, it holds that $\tau = \{i = \tau_i\}^I$ where $\tau_i : \lceil \bar{P}_i \rceil$ for all $i \in I$.

Claim: $(E \vdash^{\mathsf{i}} M : |\mathsf{select}^{\mathsf{b}}(\tau_i, \bar{A})|)$ *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 $(E \vdash^{\mathsf{i}} M : |\mathsf{select}^{\mathsf{b}}(\tau, (i, \bar{A}))|)$, by assumption. But $(\mathsf{select}^{\mathsf{b}}(\tau, (i, \bar{A})) = \mathsf{select}^{\mathsf{b}}(\tau_i, \bar{A}))$.

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

Proof of Theorem 7.4. The proof is by lexicographical induction on the pair $\langle \mathsf{size}(M), \mathsf{size}(\kappa)\rangle$. Suppose $(E \vdash^{\mathsf{B}} M : \tau \text{ at } \kappa)$ and $(\bar{A} \trianglelefteq \kappa)$.
Case, $\kappa = \{i = \kappa_i\}^I$**:** By definition of $\trianglelefteq$, there are $\bar{A}'$ and i in I such that $(\kappa = (i, \bar{A}'))$ and $(\bar{A}' \trianglelefteq \kappa_i)$. By Lemmas 7.3(1) and 6.9,

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

By Lemma 7.5, $(\mathsf{size}(\mathsf{select}^{\mathsf{b}}(M, (i, *))) = \mathsf{size}(M))$. Therefore, by induction hypotheses and Lemma 6.15,

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

Then, by Lemma 7.6,

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

Case, $\kappa = *$**:** Because $(\bar{A} \trianglelefteq *)$, 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 $(\tau \simeq \forall\bar{P}.\sigma)$ and

$$\mathsf{expand}(E, \bar{P}) \vdash^{\mathsf{B}} N : \sigma \text{ at } \lceil\bar{P}\rceil$$

By induction hypothesis,

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

Then, by Lemma 7.3(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 7.7(1).
Subcase, $M = N[\bar{A}]$**:** By inversion of the typing rule for type selection application, there is a type σ such that $(\mathsf{select}^{\mathsf{i}}(\sigma, \bar{A}) \simeq \tau)$ and $(E \vdash^{\mathsf{B}} N : \sigma \text{ at } *)$. Then, by induction hypothesis, $(|E| \vdash^{\mathsf{i}} |N| : |\sigma|)$. Then $(|E| \vdash^{\mathsf{i}} |M| : |\tau|)$, by Lemma 7.7(2). □

7.2 From Intersection Types to Branching Types

We define

$$\begin{array}{rcl} \mathsf{e} & : & \mathsf{IntTy} \to \mathsf{IndTy} \\ \mathsf{e}(\alpha) & = & \alpha \\ \mathsf{e}(\tau \to \sigma) & = & \mathsf{e}(\tau) \to \mathsf{e}(\sigma) \\ \mathsf{e}(\wedge\{i = \tau_i\}^I) & = & \forall \mathsf{join}\{i = *\}^I.\{i = \mathsf{e}(\tau_i)\}^I \end{array}$$

The function is extended to environments by $\mathsf{e}(E)(x) = \mathsf{e}(E(x))$.

Theorem 7.8. *If* $(E \vdash^{\mathsf{i}} M : \tau)$*, then there is a* λ^{B}*-term* N *such that* $|N| = M$ *and* $(\mathsf{e}(E) \vdash^{\mathsf{B}} N : \mathsf{e}(\tau) \text{ at } *)$. □

Proof. Straightforward induction on the derivation of $(E \vdash^{\mathsf{i}} M : \tau)$. □

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].

[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), 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.

[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.

[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.* 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, Paris, France, 1974. Springer-Verlag.

[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*, 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*, LNCS. Springer-Verlag, 2002.