Type Inference in Presence of Positive Subtyping with Bounded Quantification

Kathleen Fisher and Pascal-Louis Perez

June 7, 2007

Abstract

We investigate the introduction of a positive subtyping relation in a Hindley/Milner style type inference. Positive subtyping is a restricted form of the standard contra-variant subtyping and makes type inference algorithmically more efficient. By itself, positive subtyping is not as expressive as one would hope and we present a constraint-based syntax-directed type inference system targeted at improving this expressiveness. We extend this type inference system with declarative rules simplifying its presentation and give a formal comparison between our system and Hindley/Milner style type systems. In particular, we prove that subtyping can be incorporated into an existing type system whilst preserving the complexity class of type inference. We show that in our setting, satisfiability of inequalities is PTIME-complete. Finally, we use a constraint satisfaction algorithm and show its application to our type system.

Acknowledgments

All my gratitude goes to Kathleen Fisher for making this research possible. In the early steps of this research, she gave me a lot of liberty to explore potential directions and provided me with crucial insights. She was flexible enough to allow me to work remotely despite the extra workload this represented for her. During the elaboration of the type inference system, her intuition and knowledge got me out of difficult proofs and bad directions.

Thanks goes to Alex Aiken and John C. Mitchell, my two advisors, for enabling me to pursue the Distinction in Research during my Masters. I also thank Professor Mitchell for his financial support.

I am thankful to Martin Odersky and Uwe Nestmann for introducing me to programming language theory and sharing with me their passion. I also thank, in no specific order, Marcin Benke, François Pottier, Didier Rémy, John Reppy and Martin Sulzmann for their various contributions.

Last, I would also like to thank Thibault Guicherd-Callin for proof-reading this thesis alongside of his time consuming projects.

Finally, I would like to thank my wife Dalia and daughter Aline for their patience, as well as the rest of my family for making my studies at Stanford University possible.

Contents

Chapter 1

Introduction

Since its discovery, Hindley/Milner style type inference systems have been a popular approach to handling types in functional programming languages and has evolved rapidly. These type systems share totality and thus allow the programmer to omit all type annotations. This approach is challenging and many extensions have been shown intractable or undecidable (such as arbitrary-rank types [17]).

In this work, we consider the problem of type inference in the presence of positive subtyping. Subtyping captures the type safety of substitutability of an expression and arises naturally in object-oriented languages. In essence, an expression e_1 can be safely substituted for e_2, if e_1's type is a subtype of e_2's type. From an information theoretic view, subtyping can be seen as a "more informative" or "more precise" relation.

Functional subtyping considers the relation between two functional types and exhibits algebraic difficulties. Take a function f of type $\mathsf{A} \to \mathsf{B}$ used in some context E forming the expression $E[f]$. The function f may be applied to any expression with type A and produces an expression of type B. Therefore, we can substitute for f a function that produces subtypes of B since these will be safely used in the context. Similarly, we can substitute for f a function expecting supertypes of A as within the context E, the function is applied to expressions with type A. Indeed, expressions with type A are safe substitutes in a context that expects a supertype of A.

We can observe that the functional subtyping relation is co-variant, or positive, in the return type and contra-variant, or negative, in the argument type. Hoang and Mitchell [9, 10] have shown that constraint satisfaction in presence of the standard contra-variant subtyping relation can be PSPACE-hard. We study the restriction of the subtyping relation by disallowing contra-variance, and call this positive subtyping. In a positive subtyping relation, types vary only co-variantly or positively. For functional subtyping, the argument type becomes invariant and the return type stays co-variant. The positive subtyping relation is a strict subset of the standard subtyping relation.

We discuss the challenges introduced by the loss of expressiveness and use constrained type schemes $\forall \bar{\mathsf{X}} | C.\mathsf{T}$ to recover a restricted form of contra-variance. Constraints are built from subtyping constraints and conjunctions. We add for presentation and for semantic arguments the predicate true and false with the usual meaning as well as existential quantification. We interpret constraints by grounding them, making their interpretation a natural extension of the subtyping relation and logical conjunction. Type schemes' interpretation follows from the meaning of constraints.

We present a syntax-directed type inference system in the line of HM(X) [36]. An alternative declarative version is given. This alternative presentation simplifies the understanding of the algorithmic presentation whilst preserving structural properties. In addition, the declarative formulation allows a programmer to reason about the type system as if it were a standard object-oriented type system, in which constraints are present to recover expressiveness, not for typing subtleties. We give a formal comparison between our system and Hindley/Milner style type systems and show that our type system is a strict extension.

We then show that positive subtyping can be incorporated into an existing type system whilst preserving the complexity class of type inference. In fact, positive subtyping lifts the subtyping relation without altering the shape of the ordering relation. In our setting, satisfiability of inequalities is PTIME-complete.

Finally, we adapt a constraint satisfaction algorithm, developed by Benke [5] to verify the satisfiability of constraints gathered during type inference and thus verify that programs are well-typed. A prototype implementation of our system is available on `http://www.stanford.edu/~plperez/` and we give a few examples in appendix.

1.1 Expressiveness

Positive subtyping is a strict subset of the usual notion of subtyping and is therefore over-protective. It prevents safe substitutions and is especially noticeable with higher-order functions. For instance, one might need to use points as keys in a hash table and thus hash point objects. From an implementation point of view, one needs a $\mathsf{Point} \to \mathsf{Int}$ hash function, but from a client's perspective a generic hash function $\mathsf{Object} \to \mathsf{Int}$ may be sufficient. Because of positive subtyping, simple mismatch, such as the one we highlighted, prevents the typability of the program. Hofmann and Pierce [11] nonetheless note that this weaker relation retains sufficient flexibility to model objects, encapsulation and inheritance.

Formalizing the preceding example, consider a version of the apply function $\lambda x.\lambda f.f\ x$, taking a value followed by a function and applying the function to the value. With a standard contra-variant subtyping relation, the type $\forall \mathsf{X}, \mathsf{Y}.\mathsf{X} \to (\mathsf{X} \to \mathsf{Y}) \to \mathsf{Y}$ is expressive enough. If the value is a colored point c, we have $\lambda f.f\ c$ whose type is now

$$\forall \mathsf{Y}.(\mathsf{ColoredPoint} \to \mathsf{Y}) \to \mathsf{Y}$$

since X has been instantiated to $\mathsf{ColoredPoint}$. With positive subtyping, only functions whose domain is $\mathsf{ColoredPoint}$ are admissible.

The decoupling between the type derivation of apply to c and the resulting expression to a function f introduces additional difficulty. If the two applications were typed together in one derivation, one could instantiate X to f's argument type.

We can see in apply's type

$$\forall \mathsf{X}, \mathsf{Y}.\underline{\mathsf{X}} \to (\underline{\mathsf{X}} \to \mathsf{Y}) \to \mathsf{Y}$$

that the two X's are unnecessarily coupled together. A looser approach would number them X_1 and X_2 and require that $\mathsf{X}_1 <: \mathsf{X}_2$. Introducing bounded quantifiers allows having a positive subtyping relationship and yet use a restricted form of contra-variance. Continuing the previous example, we have

$$\forall \bar{\mathsf{X}}, \mathsf{Y} | \mathsf{X}_1 <: \mathsf{X}_2 . \mathsf{X}_1 \to (\mathsf{X}_2 \to \mathsf{Y}) \to \mathsf{Y}$$

which allows us to add extra liberty to cope with the loss of positive subtyping. If we now consider a higher-order case, with a length function l of type $\mathsf{Point} \to \mathsf{Int}$, we have

$$(\lambda x.\lambda f.f\ x)\ l : \forall \mathsf{Y}. ((\mathsf{Point} \to \mathsf{Int}) \to \mathsf{Y}) \to \mathsf{Y}$$

since the constraint $\mathsf{Point} \to \mathsf{Int} <: \mathsf{X}_2$ forces X_2 to be equated with $\mathsf{Point} \to \mathsf{Int}$. Here, positive subtyping leads to a restrictive type. One could transform this partially grounded type based on a variance analysis to

$$\forall \mathsf{X}, \mathsf{Y} | \mathsf{X} <: \mathsf{Point}. ((\mathsf{X} \to \mathsf{Int}) \to \mathsf{Y}) \to \mathsf{Y}$$

but we do not consider this extension in this work. Another challenge with an object-oriented type system is heterogeneous collections. Suppose we have a sequencing operator $e_1 ; e_2$ which is syntactic sugar for $\lambda x.e_2\ e_1$ for an x not appearing in e_2. We can define an apply twice function $\lambda x_1.\lambda x_2.\lambda f.f\ x_1 ; f\ x_2$ taking two values x_1, x_2 and a function f applied sequentially to the two values. The Hindley/Milner type is

$$\forall \mathsf{X}, \mathsf{Y}. \mathsf{X} \to \mathsf{X} \to (\mathsf{X} \to \mathsf{Y}) \to \mathsf{Y}$$

whose rigidity is very limiting. Instead, using bounded quantifier one might have

$$\forall \bar{\mathsf{X}}, \mathsf{Y} | \mathsf{X}_1 <: \mathsf{X}_3 \wedge \mathsf{X}_2 <: \mathsf{X}_3 . \mathsf{X}_1 \to \mathsf{X}_2 \to (\mathsf{X}_3 \to \mathsf{Y}) \to \mathsf{Y}$$

allowing independent type derivations without preventing further uses.

1.2 Row Polymorphism

Another approach adds positive subtyping to row polymorphism. Rows were introduced by Rémy [32] and are described thoroughly in Pottier and Rémy [29]. A row ΠR is an infinite support record. We denote a single labeled type by $l : \mathsf{T}$ and the infinite part of ΠR by $\delta\mathsf{T}$. A typical one-dimensional point object whose record type is $\{x : \mathsf{Int}\}$ would have a row type $\Pi(x : \mathsf{Int}, \delta\top)$, and a selection function $\lambda x.x.l$ would have a polymorphic row type $\forall \bar{\mathsf{X}}.\Pi(l : \mathsf{X}_1, \mathsf{X}_2) \to \mathsf{X}_1$ where the kind of X_2 is a partial row on $\mathcal{L} \setminus \{l\}$.

By analogy with records, we can extend the subtyping relation to rows by considering a row ΠR_1 a subtype of ΠR_2 if all types in R_1 are subtypes of R_2 with a label-wise comparison; for example $\Pi(x : \mathsf{Int}, y : \mathsf{Bool}, \delta\top)$ is a subtype of $\Pi(x : \mathsf{Int}, \delta\top)$.

This approach is more expressive than plain positive subtyping. Nonetheless, similar difficulties as the one discussed previously arise, especially with heterogeneous collections.

Chapter 2

Syntax

We consider a slightly extended version of the simply typed λ-calculus with constants, record construction and record selection. Records are introduced as a means for structural subtyping.

2.1 Expressions

The expressions we consider are generated by the grammar

$$e ::= c \mid x \mid \lambda x.e \mid e_1\ e_2 \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid \left\{\vec{l} = \vec{e_l}\right\} \mid e.l.$$

We have constants (denoted by c), variables (denoted by x), anonymous functions ($\lambda x.e$), application ($e_1\ e_2$), let polymorphism ($\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$), record expressions ($\left\{\vec{l} = \vec{e_l}\right\}$) and finally record selection ($e.l$). Record labels l are chosen from a set of labels $\mathcal{L}$, which is infinite but denumerable.

Definition 2.1.1 *The free variables* fv *of an expression e is defined inductively by*

$$\mathrm{fv}(c) = \emptyset \qquad \mathrm{fv}(x) = \{x\} \qquad \mathrm{fv}\,(\lambda x.e) = \mathrm{fv}(e) \setminus \{x\} \qquad \mathrm{fv}\,(e_1\ e_2) = \mathrm{fv}(e_1) \cup \mathrm{fv}(e_2)$$

$$\mathrm{fv}\,(\mathit{let}\ x = e_1\ \mathit{in}\ e_2) = \mathrm{fv}(e_1) \cup \mathrm{fv}(e_2) \setminus \{x\} \qquad \mathrm{fv}\left(\left\{\vec{l} = \vec{e_l}\right\}\right) = \bigcup_{l \in \vec{l}} \mathrm{fv}(e_l) \qquad \mathrm{fv}\,(e.l) = \mathrm{fv}(e)$$

2.2 Monomorphic Types

The monomorphic types we consider are

$$\mathsf{T} ::= \top \mid \mathsf{B} \mid \mathsf{X} \mid \mathsf{T}_1 \to \mathsf{T}_2 \mid \left\{\vec{l} : \vec{\mathsf{T}_l}\right\}.$$

B ranges over a set of base types (such as Int, Bool or Char). We also have type variables represented by X, functional types $\mathsf{T}_1 \to \mathsf{T}_2$ and record types $\left\{\vec{l} : \vec{\mathsf{T}_l}\right\}$. Finally, the top type $\top$ is a supertype of all types. We use T or U to range over monomorphic types. We refer to the infinite set of type variables as $\mathcal{V}$ and refer to the infinite set of monomorphic types generated by the previous grammar as $\mathcal{T}$.

Definition 2.2.1 *The free type variables* ftv *of a monomorphic type* T *is defined inductively by*

$$\mathrm{ftv}(\top) = \emptyset \qquad \mathrm{ftv}(B) = \emptyset \qquad \mathrm{ftv}(X) = \{X\} \qquad \mathrm{ftv}\left(T_1 \to T_2\right) = \mathrm{ftv}(T_1) \cup \mathrm{ftv}(T_2) \qquad \mathrm{ftv}\left(\left\{\vec{l} : \vec{T_l}\right\}\right) = \bigcup_{l \in \bar{l}} \mathrm{ftv}(T_l)$$

Definition 2.2.2 *We define the set of ground types* $\mathcal{M}$ *as*

$$\mathcal{M} = \{T \in \mathcal{T} \mid \mathrm{ftv}(T) = \emptyset\}$$

that is all types not containing any type variable. A ground type is a type contained in $\mathcal{M}$*.*

2.3 Polymorphic Types

$$\begin{array}{rcl} \mathsf{S} & ::= & \forall \bar{\mathsf{X}} | C.\mathsf{T}. \\ C & ::= & \mathsf{true} \mid \mathsf{false} \mid \mathsf{T}_1 <: \mathsf{T}_2 \mid C_1 \wedge C_2 \mid \exists \bar{\mathsf{X}}.C. \end{array}$$

We may consider T a polymorphic type using the syntactic sugar $\forall \emptyset | \mathsf{true}.\mathsf{T}$. The constraints are constructed from the two nullary predicates true and false, subtyping, conjunction and existential quantification. We add the syntactic sugar $\mathsf{T}_1 = \mathsf{T}_2$ for $\mathsf{T}_1 <: \mathsf{T}_2 \wedge \mathsf{T}_2 <: \mathsf{T}_1$, which gives a syntactic meaning to equality (theorem 2.6.2). We use C or D to range over constraints, S to range over type schemes and denote by $\mathcal{S}$ the set of type schemes.

Definition 2.3.1 *We extend the notion of free type variables to constraints with*

$$\mathrm{ftv}\,(\mathit{true}) = \emptyset \qquad \mathrm{ftv}\,(\mathit{false}) = \emptyset \qquad \mathrm{ftv}\,(T_1 = T_2) = \mathrm{ftv}\,(T_1) \cup \mathrm{ftv}\,(T_2) \qquad \mathrm{ftv}\,(T_1 <: T_2) = \mathrm{ftv}\,(T_1) \cup \mathrm{ftv}\,(T_2)$$

$$\mathrm{ftv}\,(C_1 \wedge C_2) = \mathrm{ftv}\,(C_1) \cup \mathrm{ftv}\,(C_2) \qquad \mathrm{ftv}\left(\exists \bar{X}.C\right) = \mathrm{ftv}\,(C) \setminus \bar{X}$$

and to type schemes with

$$\mathrm{ftv}\left(\forall \bar{X} | C.T\right) = \mathrm{ftv}(T) \cup \mathrm{ftv}(C) \setminus \bar{X}.$$

The definition of free type variables in constraints is syntactic. In contrast, HM(X) [36] defines free type variables as the set of variables interfering with the constraint, that is

$$\mathrm{ftv}'(C) = \{\mathsf{X} \mid \exists \mathsf{X}.C \not\equiv C\}.$$

where constraint equivalence is defined in (definition 2.7.6). For instance $\mathrm{ftv}(\mathsf{X} <: \mathsf{X}) = \{\mathsf{X}\}$ whereas $\mathrm{ftv}'(\mathsf{X} <: \mathsf{X}) = \emptyset$, and more generally $\mathrm{ftv}'(C) \subseteq \mathrm{ftv}(C)$. We choose to use a syntactic approach for simplicity, and one can see in (theorem 2.7.1) and its proof a reconciliation of the two choices. Given a constraint C, one can apply the simplifications outlined in the proof to obtain C' and we have $\mathrm{ftv}(C') = \mathrm{ftv}'(C)$.

2.4 Environment

An environment is a set of bindings $x : \mathsf{S}$, which we represent by Γ. In contrast with Hindley/Milner style type systems, our environment contains only polymorphic types. An environment can be viewed as a partial function from variables ($\mathcal{V}$) to type schemes ($\mathcal{S}$). In this regard, we may write $\Gamma(x) = \mathsf{S}$ if $x : \mathsf{S} \in \Gamma$, dom($\Gamma$) for the environment's domain and range(Γ) for its range.

2.5 Base Types

As mentioned before, a set of base types is present (such as Int, Bool or Char) with their associated constants. We have $\mathcal{B}$ which is a pseudo-environment mapping constants to their base type. For instance, $\mathcal{B}(5) = \mathsf{Int}$ and $\mathcal{B}(\mathsf{true}) = \mathsf{Bool}$.

2.6 Subtyping

$$(\text{BQ-S-TOP})\ \frac{}{\mathsf{T} <: \top} \qquad (\text{BQ-S-REF})\ \frac{}{\mathsf{T} <: \mathsf{T}}$$

$$(\text{BQ-S-FUN})\ \frac{\mathsf{T}_1 <: \mathsf{T}_2}{\mathsf{T} \to \mathsf{T}_1 <: \mathsf{T} \to \mathsf{T}_2} \qquad (\text{BQ-S-REC})\ \frac{\mathsf{T}^1_l <: \mathsf{T}^2_l\ (l \in L_2) \qquad L_2 \subseteq L_1}{\left\{l : \mathsf{T}^1_l \mid l \in L_1\right\} <: \left\{l : \mathsf{T}^2_l \mid l \in L_2\right\}}$$

The (BQ-S-TOP) rule states that the top type is a super type of every type; (BQ-S-REF) states that the subtyping relation is reflexive; the (BQ-S-FUN) rule introduces a positive subtyping for the arrow type, where the left hand side is invariant; and finally the (BQ-S-REC) rule adds width and depth record subtyping. This rule can alternatively be formulated in two separate rules.

$$(\text{BQ-S-REC-WIDTH})\ \frac{L_2 \subseteq L_1}{\left\{l : \mathsf{T}^1_l \mid l \in L_1\right\} <: \left\{l : \mathsf{T}^2_l \mid l \in L_2\right\}} \qquad (\text{BQ-S-REC-DEPTH})\ \frac{\mathsf{T}^1_l <: \mathsf{T}^2_l}{\left\{l : \mathsf{T}^1_l \mid l \in L\right\} <: \left\{l : \mathsf{T}^2_l \mid l \in L\right\}}$$

This latter formulation is equivalent to the former. We do not consider any initial subtyping relation among base types, but show later that one can relax this choice to an arbitrary relation satisfying the Helly property (definition 5.1.4).

Theorem 2.6.1 *Normal Form (for monotypes)*

1. *If $\top <: \mathsf{T}$ then $\mathsf{T} = \top$.*

2. *If $\mathsf{B} <: \mathsf{T}$ then $\mathsf{T} = \mathsf{B}$ or $\mathsf{T} = \top$, if $\mathsf{T} <: \mathsf{B}$ then $\mathsf{T} = \mathsf{B}$.*

3. *If $\mathsf{X} <: \mathsf{T}$ then $\mathsf{T} = \mathsf{X}$ or $\mathsf{T} = \top$, if $\mathsf{T} <: \mathsf{X}$ then $\mathsf{T} = \mathsf{X}$.*

4. *If $\mathsf{T}_1 \to \mathsf{T}_2 <: \mathsf{U}$ then $\mathsf{U} = \mathsf{T}_1 \to \mathsf{T}_3$ and $\mathsf{T}_2 <: \mathsf{T}_3$, or $\mathsf{U} = \top$; if $\mathsf{U} <: \mathsf{T}_1 \to \mathsf{T}_2$ then $\mathsf{U} = \mathsf{T}_1 \to \mathsf{T}_3$ and $\mathsf{T}_3 <: \mathsf{T}_2$.*

5. *If* $\left\{l : T^1_l \mid l \in L_1\right\} <: U$ *then* $U = \left\{l : T^2_l \mid l \in L_2\right\}$ *and* $T^1_l <: T^2_l\ (l \in L_2)$, $L_2 \subseteq L_1$, *or* $U = \top$; *if* $U <: \left\{l : T^1_l \mid l \in L_1\right\}$ *then* $U = \left\{l : T^2_l \mid l \in L_2\right\}$ *and* $T^2_l <: T^1_l\ (l \in L_1)$, $L_1 \subseteq L_2$.

Proof (theorem 2.6.1) By rule inspection and the absence of subtyping relations among base types.

Theorem 2.6.2 *The subytping relation on monotypes is reflexive, transitive and antisymmetric.*

Proof (theorem 2.6.2) The subtyping relation is reflexive by definition.

Suppose $T_1 <: T_2$ and $T_2 <: T_3$. We will show that the subtyping relation is transitive by induction on T_3.
$\boxed{\top}$ Applying (BQ-s-top) yields $T_1 <: \top = T_3$.

$\boxed{B \text{ (or } X)}$ By normal form, T_2 is equal to B (respectively X) and so is T_1. By reflexivity $T_1 <: T_3$.

$\boxed{T_{31} \to T_{32}}$ By normal form

$$T_2 = T_{21} \to T_{22} \quad \text{therefore} \quad T_1 = T_{11} \to T_{12}$$

Since $T_2 <: T_3$, we have $T_{21} = T_{31}$ and by $T_1 <: T_2$ we have $T_{11} = T_{21}$. By induction $T_{12} <: T_{32}$. Applying (BQ-s-fun) yields $T_1 <: T_3$.

$\boxed{\left\{\vec{l_3} : \vec{T_3}\right\}}$ By normal form

$$T_2 = \left\{\vec{l_2} : \vec{T_2}\right\} \quad \text{therefore} \quad T_1 = \left\{\vec{l_1} : \vec{T_1}\right\}$$

and $\bar{l}_3 \subseteq \bar{l}_2 \subseteq \bar{l}_1$, $T^l_2 <: T^l_3\ (l \in \bar{l}_3)$ and $T^l_1 <: T^l_2\ (l \in \bar{l}_2)$. By induction $T^l_1 <: T^l_3\ (l \in \bar{l}_3)$ and applying (BQ-s-rec) yields $T_1 <: T_3$.

Now, suppose $T_1 <: T_2$ and $T_2 <: T_1$. We will show that the subtyping relation is antisymmetric by induction on T_1.
$\boxed{\top}$ By normal form $T_2 = \top$.

$\boxed{B \text{ (or } X)}$ By normal form $T_2 = B$ (respectively X) and only (BQ-s-ref) applies, thus $T_1 = T_2$.

$\boxed{T_{11} \to T_{12}}$ By normal form on $T_2 <: T_1$ we have $T_2 = T_{21} \to T_{22}$ therefore $T_{11} = T_{21}$ and by induction $T_{12} = T_{22}$ thus $T_1 = T_2$.

$\boxed{\left\{\vec{l_1} : \vec{T_1}\right\}}$ By normal form on $T_2 <: T_1$ then on $T_1 <: T_2$

$$T_2 = \left\{\vec{l_2} : \vec{T_2}\right\} \quad \text{and} \quad \bar{l}_1 = \bar{l}_2$$

(since T_1 is a record) and by induction $T^l_1 = T^l_2$, thus $T_1 = T_2$.

Definition 2.6.1 *The upward closure of a monotype T, written $\uparrow T$ is defined as $\{U \in \mathcal{T} \mid T <: U\}$ and is extended to a set of monotypes T as*

$$\uparrow T = \bigcup_{T \in T} \uparrow T.$$

Likewise, we define the downward closure $\downarrow T$ as $\{U \in \mathcal{T} \mid U <: T\}$ and extend it to a set of monotypes similarly $\downarrow T = \bigcup_{T \in T} \downarrow T$.

Lemma 2.6.1 *$T \in \uparrow T$ and $T \in \downarrow T$*

Proof (lemma 2.6.1) By reflexivity of the subtyping relation (theorem 2.6.2).

Theorem 2.6.3 *$T_1 <: T_2$ if and only if $\uparrow T_2 \subseteq \uparrow T_1$.*

Proof (theorem 2.6.3) By double implication.
$\boxed{\Rightarrow}$ Suppose $\mathsf{T}_1 <: \mathsf{T}_2$. Take $\mathsf{T}_3 \in \uparrow \mathsf{T}_2$, by definition $\mathsf{T}_2 <: \mathsf{T}_3$. By transitivity (theorem 2.6.2) $\mathsf{T}_1 <: \mathsf{T}_3$ and therefore $\mathsf{T}_3 \in \uparrow \mathsf{T}_1$, thus $\uparrow \mathsf{T}_2 \subseteq \uparrow \mathsf{T}_1$.

$\boxed{\Leftarrow}$ Suppose $\uparrow \mathsf{T}_2 \subseteq \uparrow \mathsf{T}_1$. Since $\mathsf{T}_2 \in \uparrow \mathsf{T}_2$ (lemma 2.6.1) we have $\mathsf{T}_2 \in \uparrow \mathsf{T}_1$ and by definition $\mathsf{T}_1 <: \mathsf{T}_2$.

Theorem 2.6.4 *$T_1 <: T_2$ if and only if $\downarrow T_1 \subseteq \downarrow T_2$.*

Proof (theorem 2.6.4) By double implication.
$\boxed{\Rightarrow}$ Suppose $\mathsf{T}_1 <: \mathsf{T}_2$. Take $\mathsf{T}_3 \in \downarrow \mathsf{T}_1$, by definition $\mathsf{T}_3 <: \mathsf{T}_1$. By transitivity (theorem 2.6.2) $\mathsf{T}_3 <: \mathsf{T}_2$ and therefore $\mathsf{T}_3 \in \downarrow \mathsf{T}_2$, thus $\downarrow \mathsf{T}_1 \subseteq \downarrow \mathsf{T}_2$.

$\boxed{\Leftarrow}$ Suppose $\downarrow \mathsf{T}_1 \subseteq \downarrow \mathsf{T}_2$. Since $\mathsf{T}_1 \in \downarrow \mathsf{T}_1$ (lemma 2.6.1) we have $\mathsf{T}_1 \in \downarrow \mathsf{T}_2$ and by definition $\mathsf{T}_1 <: \mathsf{T}_2$.

2.7 Meaning of Constraints

In this section, we give an interpretation to constraints. We have adapted the interpretation introduced by Pottier and Rémy [28, 29] for our purpose.

Definition 2.7.1 *A map ψ is a total mapping from type variables $\mathcal{V}$ to types $\mathcal{T}$. Maps are extended to types by $\phi(P\ T_1\ \ldots\ T_n) = P\ \phi(T_1)\ \ldots\ \phi(T_n)$, where P is an n-ary type constructor.*

Base types such as Int are nullary type constructors. Thus, thanks to the extension of maps to type constructors we have $\psi(\mathsf{Int}) = \mathsf{Int}$.

Definition 2.7.2 *A ground assignment ϕ is a map whose range is restricted to ground types $\mathcal{M}$.*

Definition 2.7.3 *A substitution $\theta_{\bar{X}}$ is a mapping from $\bar{X} \subseteq \mathcal{V}$ to types $\mathcal{T}$ and a ground substitution $\theta_{\bar{X}}$ is a substitution whose range is restricted to ground types $\mathcal{M}$.*

One can see a ground assignment as a ground substitution on $\mathcal{V}$ and vice-versa. The meaning of a constraint C under a ground assignment ϕ, written $\phi \models C$ is defined by the following rules.

$$\text{(BQ-c-true)}\ \frac{}{\phi \models \mathsf{true}} \qquad \text{(BQ-c-sub)}\ \frac{\phi(\mathsf{T}_1) <: \phi(\mathsf{T}_2)}{\phi \models \mathsf{T}_1 <: \mathsf{T}_2}$$

$$\text{(BQ-c-and)}\ \frac{\phi \models C_1 \qquad \phi \models C_2}{\phi \models C_1 \wedge C_2} \qquad \text{(BQ-c-exists)}\ \frac{\phi \models \theta_{\bar{X}}(C)}{\phi \models \exists \bar{X}.C}$$

If $\phi \models C$ we say that the ground assignment ϕ satifies C, or that C is satisfied by the ground assignment ϕ. The lack of interpretation for the false predicate captures its inherent unsatisfiability.

Definition 2.7.4 *We say that C_1 implies C_2, written $C_1 \Vdash C_2$, if and only if for all ground assignments ϕ, $\phi \models C_1$ implies $\phi \models C_2$.*

Definition 2.7.5 *We say that a constraint C is satisfiable, written $\models C$, if and only if* true $\Vdash \exists\, \mathrm{ftv}(C).C$.

Definition 2.7.6 *We say that C_1 is equivalent to C_2, written $C_1 \equiv C_2$, if and only if $C_1 \Vdash C_2$ and $C_2 \Vdash C_1$.*

Theorem 2.7.1 *If $C_1 \Vdash C_2$ then there exists a constraint C_2' such that $C_2 \equiv C_2'$ and* $\mathrm{ftv}(C_2') \subseteq \mathrm{ftv}(C_1)$.

Proof (theorem 2.7.1) By induction on the structure of C_2.
$\boxed{\mathsf{true}}$ Trivial.

$\boxed{\mathsf{T}_1 <: \mathsf{T}_2}$ Suppose $C_1 \Vdash \mathsf{T}_1 = \mathsf{T}_2$. If $\mathrm{ftv}\,(\mathsf{T}_1 = \mathsf{T}_2)$ is a subset of $\mathrm{ftv}(C_1)$, nothing needs to be proved. Thereon we suppose that it is not the case. Take

$$\bar{\mathsf{X}} = \mathrm{ftv}\,(\mathsf{T}_1 = \mathsf{T}_2) \setminus \mathrm{ftv}(C_1)$$

and let ϕ be a ground assignment satisfying C_1. By definition (definition 2.7.4) we have $\phi \models \mathsf{T}_1 = \mathsf{T}_2$. We show by induction on T_2 that the constraint $\mathsf{T}_1 = \mathsf{T}_2$ is equivalent to some C_2' with $\mathrm{ftv}(C_2') \subseteq \mathrm{ftv}(C_1)$.
[$\top$] Since any type is a subtype of the top type, C_2 is equivalent to true which does not contain any free type variables, concluding this case.

[B] By normal form (theorem 2.6.1), $\phi(\mathsf{T}_1) = \mathsf{B}$. Therefore T_1 is either be B or a type variable X and $\bar{\mathsf{X}} = \{\mathsf{X}\}$. In the first case, we are done and we show that second leads to a contradiction. Take a ground assignment ϕ' equal to ϕ everywhere but in X such that $\phi'(\mathsf{X}) \neq \mathsf{B}$. By construction ϕ' satisfies C_1 but not C_2 leading to a contradiction, concluding this case.

$[\mathsf{U} \to \mathsf{T}_2']$ By normal form (theorem 2.6.1), $\phi(\mathsf{T}_1) = \mathsf{U} \to \mathsf{T}_1'$ and $\mathsf{T}_1' <: \mathsf{T}_2'$. Therefore, C_2 is equivalent to $\mathsf{T}_1' <: \mathsf{T}_2'$ which by induction has an equivalent satisfying the theorem.

$[\left\{l_{2_1} : \mathsf{T}_2^{l_{2_1}}, \ldots, l_{2_n} : \mathsf{T}_2^{l_{2_n}}\right\}]$ Exact same reasoning as for the previous case, C_2 is equivalent to

$$\bigwedge_{l \in l_2} \mathsf{T}_1^l <: \mathsf{T}_2^l$$

and by induction all sub-constraints have equivalent constraints satisfying the theorem.

$\boxed{C_2' \wedge C_2''}$ Suppose that $C_1 \Vdash C_2' \wedge C_2''$, then $C_1 \Vdash C_2'$ and $C_1 \Vdash C_2''$. By induction, C_2' is equivalent to C_3' such that $\mathrm{ftv}(C_3') \subseteq \mathrm{ftv}(C_1)$ and C_2'' is equivalent to C_3'' such that $\mathrm{ftv}(C_3'') \subseteq \mathrm{ftv}(C_1)$. Therefore $C_2' \wedge C_2''$ is equivalent to $C_3' \wedge C_3''$ and

$$\mathrm{ftv}\left(C_3' \wedge C_3''\right) = \mathrm{ftv}(C_3') \cup \mathrm{ftv}(C_3'') \subseteq \mathrm{ftv}(C_1)$$

which concludes this case.

$\boxed{\exists \bar{\mathsf{X}}.C_2'}$ Suppose that $C_1 \Vdash \exists \bar{\mathsf{X}}.C_2'$. Since the variables $\bar{\mathsf{X}}$ are bound in C_2' and there are finitely many free type variables in C_1, we can choose a non conflicting set $\bar{\mathsf{X}}$ with $\mathrm{ftv}(C_1)$. By definition (definition 2.7.4) for all ground assignments $\phi \models C_1$ implies $\phi \models \exists \bar{\mathsf{X}}.C_2'$. Fix ϕ such that $\phi \models C_1$, by interpretation we have $\phi \models \theta_{\bar{\mathsf{X}}}(C_2')$. Therefore

$$\phi \models \theta_{\bar{\mathsf{X}}}(C_1)$$

since we chose non conflicting $\bar{\mathsf{X}}$. By induction, there exists a constraint C_2'' equivalent to C_2' satisfying $\mathrm{ftv}(C_2'') \subseteq \mathrm{ftv}(C_1)$. Thus, $\exists \bar{\mathsf{X}}.C_2''$ is equivalent to $\exists \bar{\mathsf{X}}.C_2'$ and $\mathrm{ftv}\left(\exists \bar{\mathsf{X}}.C_2''\right) \subseteq \mathrm{ftv}(C_1)$, concluding this case.

2.8 Interpretation of Type Schemes

In this section, we give a semantic meaning to type schemes. The interpretation (definition 2.8.1) is adapted from Pottier and Rémy [29]. We define an equivalence and subtyping relation on type schemes and give key properties.

Definition 2.8.1 *The interpretation of a type scheme $\forall \bar{X} | C.T$ within a ground assignment ϕ is defined by*

$$[\![\forall \bar{X} | C.T]\!]_\phi = \uparrow \left\{\phi\left(\theta_{\bar{X}}(T)\right) \,\middle|\, \phi \models \theta_{\bar{X}}(C)\right\}$$

where the upward closure is defined in (definition 2.6.1).

The interpretation of a type scheme captures the set of ground types that are an instance of the type scheme. This set is upward closed to match our intuition and type scheme comparison can be made very naturally.

Intuitively, two type schemes are equivalent when their interpretation is the same. For instance, for any ground assignment ϕ we have $[\![\forall \mathsf{X}|\mathsf{Int} <: \mathsf{X}.\mathsf{X}]\!]_\phi$ equals to $\uparrow\{\mathsf{Int}, \top\}$, which in turn is equal to $\{\mathsf{Int}, \top\}$. Similarly, $[\![\forall \emptyset|\mathsf{true}.\mathsf{Int}]\!]_\phi = \uparrow\{\mathsf{Int}\} = \{\mathsf{Int}, \top\}$ indicating that the two type schemes $\forall \mathsf{X}|\mathsf{Int} <: \mathsf{X}.\mathsf{X}$ and $\forall \emptyset|\mathsf{true}.\mathsf{Int}$ are equivalent. In addition, it is always the case that $[\![\mathsf{T}]\!]_\phi = \uparrow \mathsf{T}$.

We now formalize the equivalence relation on type schemes informally outlined earlier in the next definition.

Definition 2.8.2 *We say that two type schemes S_1 and S_2 are equivalent, written $S_1 \equiv S_2$, if their interpretation is the same, i.e. $[\![S_1]\!]_\phi = [\![S_2]\!]_\phi$ for all ϕ.*

Lemma 2.8.1 *The $\equiv$ relation on type schemes is an equivalence relation.*

Proof (lemma 2.8.1) Follows from the definition of type scheme interpretation (definition 2.8.1) and the properties of set equality.

Definition 2.8.3 *We say that a type scheme S_1 is a subtype of another type scheme S_2, written $S_1 <: S_2$, if $[\![S_2]\!]_\phi \subseteq [\![S_1]\!]_\phi$ for all ϕ and we extend the notion of subtyping to environments:*

$$\text{(BQ-S-ENV)} \quad \frac{\Gamma'(x) <: \Gamma(x) \quad (x \in \operatorname{dom}\Gamma)}{\Gamma' <: \Gamma}$$

Lemma 2.8.2 *The subtyping relation on type schemes is reflexive and transitive (it is not syntactically antisymmetric).*

Proof (lemma 2.8.2) Follows from the definition of type scheme interpretation (definition 2.8.1) and the properties of the subset relation. A counter-example to syntactic antisymmetry has been given previously, i.e. $\forall \mathsf{X}|\mathsf{Int} <: \mathsf{X}.\mathsf{X}$ and $\forall \emptyset|\mathsf{true}.\mathsf{Int}$.

Lemma 2.8.3 *The subtyping relation on type schemes is antisymmetric with respect to the equivalence relation.*

Proof (lemma 2.8.3) Suppose $\mathsf{S}_1 <: \mathsf{S}_2$ and $\mathsf{S}_2 <: \mathsf{S}_1$. By definition (definition 2.8.3), for all ground assignments ϕ we have $[\![\mathsf{S}_2]\!]_\phi \subseteq [\![\mathsf{S}_1]\!]_\phi$ and $[\![\mathsf{S}_1]\!]_\phi \subseteq [\![\mathsf{S}_2]\!]_\phi$. By double inclusion $[\![\mathsf{S}_1]\!]_\phi = [\![\mathsf{S}_2]\!]_\phi$ and by (definition 2.8.2) $\mathsf{S}_1 \equiv \mathsf{S}_2$.

Lemma 2.8.4 *If a type scheme S has no free type variables, then its interpretation is independent of any ground assignment, i.e. for all ground assignments ϕ_1, ϕ_2 we have $[\![S]\!]_{\phi_1} = [\![S]\!]_{\phi_2}$.*

Proof (lemma 2.8.4) By definition (definition 2.8.1).

The following lemma indicates that a type scheme binding more variables is always more precise than a similar type scheme binding fewer variables. In addition, unbound variables may be freely substituted.

Lemma 2.8.5 *If $\bar{X} \subseteq \bar{Y}$ then $\forall \bar{Y}|C.T <: \forall \bar{X}|\theta_{\bar{Y}\setminus\bar{X}}(C).\theta_{\bar{Y}\setminus\bar{X}}(T)$.*

Proof (lemma 2.8.5) Let $\bar{\mathsf{X}} \subseteq \bar{\mathsf{Y}}$ and take $\mathsf{T}^\star \in [\![\forall\bar{\mathsf{X}}|C.\theta_{\bar{\mathsf{Y}}\setminus\bar{\mathsf{X}}}(\mathsf{T})]\!]_\phi$. By definition (definition 2.8.1), for all ground assignments ϕ we have

$$\phi \models \theta_{\bar{\mathsf{X}}}(\theta_{\bar{\mathsf{Y}}\setminus\bar{\mathsf{X}}}(C)) \quad \text{and} \quad \phi(\theta_{\bar{\mathsf{X}}}(\theta_{\bar{\mathsf{Y}}\setminus\bar{\mathsf{X}}}(\mathsf{T}))) <: \mathsf{T}^\star$$

and we can merge $\theta_{\bar{\mathsf{X}}}$ and $\theta_{\bar{\mathsf{Y}}\setminus\bar{\mathsf{X}}}$, resulting in a substitution $\theta_{\bar{\mathsf{Y}}}$. By construction

$$\phi \models \theta_{\bar{\mathsf{Y}}}(C) \quad \text{and} \quad \phi(\theta_{\bar{\mathsf{Y}}}(\mathsf{T})) <: \mathsf{T}^\star$$

and therefore $\mathsf{T}^\star \in [\![\forall\bar{\mathsf{Y}}|C.\mathsf{T}]\!]_\phi$. Thus, $[\![\forall\bar{\mathsf{X}}|C.\theta_{\bar{\mathsf{Y}}\setminus\bar{\mathsf{X}}}]\!]_\phi \subseteq [\![\forall\bar{\mathsf{Y}}|C.\mathsf{T}]\!]_\phi$ and by definition (definition 2.8.3) $\forall\bar{\mathsf{Y}}|C.\mathsf{T} <: \forall\bar{\mathsf{X}}|C.\theta_{\bar{\mathsf{Y}}\setminus\bar{\mathsf{X}}}(\mathsf{T})$.

The following lemma follows our intuition that a type scheme binding variables vacuously is equivalent to the type scheme binding only the variables mentioned, i.e. vacuous binding can be removed.

Lemma 2.8.6 *$\forall\bar{X}|C.T \equiv \forall\bar{Y}|C.T$ where $\bar{Y} = (\text{ftv}(T) \cup \text{ftv}(C)) \cap \bar{X}$.*

Proof (lemma 2.8.6) Take $\mathsf{T}^\star \in [\![\forall\bar{\mathsf{X}}|C.\mathsf{T}]\!]_\phi$, by definition (definition 2.8.1), for all ground assignments ϕ we have

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \quad \text{and} \quad \phi(\theta_{\bar{\mathsf{X}}}(\mathsf{T})) <: \mathsf{T}^\star.$$

Let $\bar{\mathsf{Y}} = (\text{ftv}(\mathsf{T}) \cup \text{ftv}(C)) \cap \bar{\mathsf{X}}$, then $\theta_{\bar{\mathsf{X}}}(C) = \theta_{\bar{\mathsf{Y}}}(C)$ and $\theta_{\bar{\mathsf{X}}}(\mathsf{T}) = \theta_{\bar{\mathsf{Y}}}(\mathsf{T})$. Therefore

$$\phi \models \theta_{\bar{\mathsf{Y}}}(C) \quad \text{and} \quad \phi(\theta_{\bar{\mathsf{Y}}}(\mathsf{T})) <: \mathsf{T}^\star$$

thus $\mathsf{T}^\star \in [\![\forall\bar{\mathsf{Y}}|C.\mathsf{T}]\!]_\phi$, leading us to $[\![\forall\bar{\mathsf{X}}|C.\mathsf{T}]\!]_\phi \subseteq [\![\forall\bar{\mathsf{Y}}|C.\mathsf{T}]\!]_\phi$ and by definition (definition 2.8.3) $\forall\bar{\mathsf{Y}}|C.\mathsf{T} <: \forall\bar{\mathsf{X}}|C.\mathsf{T}$. In addition $\bar{\mathsf{Y}} \subseteq \bar{\mathsf{X}}$ and by lemma (lemma 2.8.5) $\forall\bar{\mathsf{X}}|C.\mathsf{T} <: \forall\bar{\mathsf{Y}}|C.\mathsf{T}$. We can conclude using the antisymmetry of the subtyping relation with respect to the equivalence relation (lemma 2.8.3) $\forall\bar{\mathsf{X}}|C.\mathsf{T} \equiv \forall\bar{\mathsf{Y}}|C.\mathsf{T}$.

The next lemma formalizes that a weaker constraint results in a more general type scheme.

Lemma 2.8.7 *If $C \mid\!\models D$ then $\forall\bar{X}|D.T <: \forall\bar{X}|C.T$.*

Proof (lemma 2.8.7) Let $C \mid\!\models D$ and take $\mathsf{T}^\star \in [\![\forall\bar{\mathsf{X}}|C.\mathsf{T}]\!]_\phi$. By definition (definition 2.8.1), for all ground assignments ϕ we have

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \quad \text{and} \quad \phi(\theta_{\bar{\mathsf{X}}}(\mathsf{T})) <: \mathsf{T}^\star.$$

Take ϕ' equal to ϕ in all positions but such that $\phi'(\mathsf{X}) = \theta_{\bar{\mathsf{X}}}(C)$ for $\mathsf{X} \in \bar{\mathsf{X}}$. By construction $\phi' \models C$ and by definition (definition 2.7.4) $\phi' \models D$. Therefore $\phi \models \theta_{\bar{\mathsf{X}}}(D)$ which allows us to conclude that $\mathsf{T}^\star \in [\![\forall\bar{\mathsf{X}}|D.\mathsf{T}]\!]_\phi$. Thus, $[\![\forall\bar{\mathsf{X}}|C.\mathsf{T}]\!]_\phi \subseteq [\![\forall\bar{\mathsf{X}}|D.\mathsf{T}]\!]_\phi$ and by definition (definition 2.8.3) $\forall\bar{\mathsf{X}}|D.\mathsf{T} <: \forall\bar{\mathsf{X}}|C.\mathsf{T}$.

Corollary 2.8.1 *$\forall\bar{X}|C_1.T \equiv \forall\bar{X}|C_2.T$ provided that $C_1 \equiv C_2$.*

Proof (corollary 2.8.1) Let $C_1 \equiv C_2$, by definition (definition 2.7.6) $C_1 \mid\!\models C_2$ and $C_2 \mid\!\models C_1$. By lemma (lemma 2.8.7) $\forall\bar{\mathsf{X}}|C_2.\mathsf{T} <: \forall\bar{\mathsf{X}}|C_1.\mathsf{T}$ and $\forall\bar{\mathsf{X}}|C_1.\mathsf{T} <: \forall\bar{\mathsf{X}}|C_2.\mathsf{T}$. Finally, by antisymmetry with respect to the equivalence relation (lemma 2.8.3) we can conclude $\forall\bar{\mathsf{X}}|C_1.\mathsf{T} \equiv \forall\bar{\mathsf{X}}|C_2.\mathsf{T}$.

The normal form theorem on type schemes unifies all the lemmas previously shown into a simple equivalence between subtyping and constraint satisfaction. One could use this theorem to derive the previous lemmas, but our direct proofs highlight the reasoning used to work with type schemes well.

Theorem 2.8.1 *Normal Form (for type schemes)* $\forall\bar{\mathsf{Y}}|C'.T' <: \forall\bar{\mathsf{X}}|C.T$ *if and only if for all ground assignments* ϕ *and all ground substitutions* $\theta_{\bar{\mathsf{X}}}$*, there exists a ground substitution* $\theta_{\bar{\mathsf{Y}}}$ *such that*

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \implies \phi \models \theta_{\bar{\mathsf{Y}}}(C') \textit{ and } \phi \models \theta_{\bar{\mathsf{Y}}}(T') <: \theta_{\bar{\mathsf{X}}}(T).$$

Proof (theorem 2.8.1) By double implication.
$\boxed{\Rightarrow}$ Let $\forall\bar{\mathsf{Y}}|C'.\mathsf{T}' <: \forall\bar{\mathsf{X}}|C.\mathsf{T}$. By definition (definition 2.8.1) and (definition 2.8.3), for all ground assignments ϕ we have

$$\uparrow\underbrace{\left\{\phi\left(\theta_{\bar{\mathsf{X}}}(\mathsf{T})\right) \,\middle|\, \phi \models \theta_{\bar{\mathsf{X}}}(C)\right\}}_{S_2} \subseteq \uparrow\underbrace{\left\{\phi\left(\theta_{\bar{\mathsf{Y}}}(\mathsf{T}')\right) \,\middle|\, \phi \models \theta_{\bar{\mathsf{Y}}}(C')\right\}}_{S_1}.$$

If S_2 is the empty set, the constraint C is not satisfiable and the theorem trivially holds. We suppose thereon that it is non-empty. Fix a ground assignment ϕ and take a ground substitution $\theta_{\bar{\mathsf{X}}}$ such that $\theta_{\bar{\mathsf{X}}}(C)$ is satisfied. In this case, we have $\theta_{\bar{\mathsf{X}}}(\mathsf{T}) \in S_2$ and by set containment $\theta_{\bar{\mathsf{X}}}(\mathsf{T}) \in \uparrow S_1$. Therefore, for some ground substitution $\theta_{\bar{\mathsf{Y}}}$

$$\phi \models \theta_{\bar{\mathsf{Y}}}(C') \quad \text{and} \quad \phi\left(\theta_{\bar{\mathsf{Y}}}(\mathsf{T}')\right) <: \phi\left(\theta_{\bar{\mathsf{X}}}(\mathsf{T})\right)$$

and applying (BQ-c-sub) yields $\phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T})$, concluding this case.

$\boxed{\Leftarrow}$ Suppose that for all ground assignments ϕ and all ground substitutions $\theta_{\bar{\mathsf{X}}}$, there exists a ground substitution $\theta_{\bar{\mathsf{Y}}}$ satisfying

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \implies \phi \models \theta_{\bar{\mathsf{Y}}}(C') \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T}).$$

Take $\mathsf{T}^\star \in [\![\forall\bar{\mathsf{X}}|C.\mathsf{T}]\!]_\phi$, by definition (definition 2.8.1)

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \quad \text{and} \quad \phi(\theta_{\bar{\mathsf{X}}}(\mathsf{T})) <: \mathsf{T}^\star$$

and by implication

$$\phi \models \theta_{\bar{\mathsf{Y}}}(C') \quad \text{and} \quad \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T})$$

therefore $\phi(\theta_{\bar{\mathsf{Y}}}(\mathsf{T}')) <: \mathsf{T}^\star$ which proves that $\mathsf{T}^\star \in [\![\forall\bar{\mathsf{Y}}|C'.\mathsf{T}']\!]_\phi$. By definition (definition 2.8.3) $\forall\bar{\mathsf{Y}}|C'.\mathsf{T}' <: \forall\bar{\mathsf{X}}|C.\mathsf{T}$ which concludes this case.

Chapter 3

Type Inference

3.1 Typing Rules

3.1.1 Syntax Directed Rules

We introduce two judgment forms. The polymorphic typing judgment is a three-place predicate with an environment, an expression and a type scheme written $\Gamma \vdash_{\mathsf{p}} e : \mathsf{S}$. The monomorphic typing judgment is a four-place predicate with a constraint, an environment, an expression and a monomorphic type written $C; \Gamma \vdash_{\mathsf{m}} e : \mathsf{T}$. The monomorphic typing judgment is in the spirit of Aiken/Wimmers and Sulzmann [2, 36].

The monomorphic typing judgment can be thought of as internal derivation, whereas the polymorphic typing judgment is the exposed portion of the type system. When implemented, optional type annotations provided by the programmer would only be given and verified at the polymorphic level.

$$\text{(BQ-SD-FIN)}\ \frac{C; \Gamma \vdash_{\mathsf{m}} e : \mathsf{T} \qquad \bar{\mathsf{X}} = \mathrm{ftv}(\mathsf{T}) \cup \mathrm{ftv}(C) \setminus \mathrm{ftv}(\Gamma) \qquad \models C}{\Gamma \vdash_{\mathsf{p}} e : \forall \bar{\mathsf{X}} | C.\mathsf{T}} \qquad \text{(BQ-SD-START)}\ \frac{\Gamma \vdash_{\mathsf{p}} e : \forall \bar{\mathsf{X}} | C.\mathsf{T}}{C; \Gamma \vdash_{\mathsf{m}} e : \mathsf{T}}$$

$$\text{(BQ-SD-BASE)}\ \frac{\mathscr{B}(c) = \mathsf{T}_c}{\Gamma \vdash_{\mathsf{p}} c : \forall \emptyset | \mathsf{true}.\mathsf{T}_c} \qquad \text{(BQ-SD-VAR)}\ \frac{\Gamma(x) = \mathsf{S}}{\Gamma \vdash_{\mathsf{p}} x : \mathsf{S}} \qquad \text{(BQ-SD-LET)}\ \frac{\Gamma \vdash_{\mathsf{p}} e_1 : \mathsf{S} \qquad C; \Gamma \cdot x : \mathsf{S} \vdash_{\mathsf{m}} e_2 : \mathsf{T}}{C; \Gamma \vdash_{\mathsf{m}} \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \mathsf{T}}$$

$$\text{(BQ-SD-REC)}\ \frac{C_1; \Gamma \vdash_{\mathsf{m}} e_1 : \mathsf{T}_1 \qquad \cdots \qquad C_n; \Gamma \vdash_{\mathsf{m}} e_n : \mathsf{T}_n}{C_1 \wedge \cdots \wedge C_n; \Gamma \vdash_{\mathsf{m}} \{l_1 = e_1, \ldots, l_n = e_n\} : \{l_1 : \mathsf{T}_1, \ldots, l_n : \mathsf{T}_n\}}$$

$$\text{(BQ-SD-SEL)}\ \frac{C; \Gamma \vdash_{\mathsf{m}} e : \mathsf{T} \qquad \mathsf{X}\ \text{is fresh}}{C \wedge \mathsf{T} <: \{l : \mathsf{X}\}; \Gamma \vdash_{\mathsf{m}} e.l : \mathsf{X}} \qquad \text{(BQ-SD-ABS)}\ \frac{C; \Gamma \cdot x : \forall \emptyset | \mathsf{true}.\mathsf{X} \vdash_{\mathsf{m}} e : \mathsf{T}}{C; \Gamma \vdash_{\mathsf{m}} \lambda x.e : \mathsf{X} \to \mathsf{T}}$$

$$\text{(BQ-SD-APP)}\ \frac{C_1; \Gamma \vdash_{\mathsf{m}} e_1 : \mathsf{T}_1 \qquad C_2; \Gamma \vdash_{\mathsf{m}} e_2 : \mathsf{T}_2 \qquad \mathsf{X}, \mathsf{Y}\ \text{are fresh}}{C_1 \wedge C_2 \wedge \mathsf{T}_1 <: \mathsf{X} \to \mathsf{Y} \wedge \mathsf{T}_2 <: \mathsf{X}; \Gamma \vdash_{\mathsf{m}} e_1\ e_2 : \mathsf{Y}}$$

The environment flows upward in the derivation and the types and constraints are synthesized, and therefore flow downward. One can notice that all 'initial' constraints are true, (BQ-SD-BASE) and (BQ-SD-ABS).

We first discuss the monomorphic typing rules. The base rule, (BQ-SD-BASE), retrieves from the pseudo-environment a constant's type and the variable rule, (BQ-SD-VAR), retrieves from the environment a variable's type. The let rule, (BQ-SD-LET), is the equivalent of Hindley/Milner's let-polymorphism. It derives e_1's polymorphic type and uses this inferred type when typing e_2. The record rule, (BQ-SD-REC), combines all constraints from the independent derivations for e_1 through e_n and gives a type to the record expression. If $n = 0$, the rule simplifies to

$$(\text{BQ-SD-REC})\ \frac{}{\mathsf{true}; \Gamma \vdash_{\mathsf{m}} \{\} : \{\}}$$

stating that the empty record has the empty record type regardless of the environment. In Aiken and Wimmers [2] this rule is expressed generically on type constructors and assigns a fresh type variable to the record expression, whilst constraining it to be a super type of the record type.

The selection rule, (BQ-SD-SEL), constrains e's inferred type using a fresh type variable. If the inferred type was a record, one could simplify the rule to extract the label's type instead of adding a constraint. By doing so, we reduce the leeway that the constraint provides, yet the preciseness of the type inference algorithm is untouched.

The abstraction rule, (BQ-SD-ABS), types anonymous functions. Because of its syntax-directed nature, one must use a type variable as the type of the bound variable x. The verbose syntax $\forall\emptyset|\mathsf{true}.\mathsf{X}$ is due to the environment binding type schemes. For clarity, we do not use syntactic sugar here.

Finally, the application rule, (BQ-SD-APP), types e_1 and e_2 and constrains their types for soundness. In an earlier version, e_1's type T_1 was equated to $\mathsf{X} \to \mathsf{Y}$. The present version is closer to (BQ-SD-SEL) and Aiken/Wimmers [2] but does not add any expressiveness. We do not require that X not appear in the free type variables of Γ. In practice, it is desirable not to select an appearing variable but it does not change the type system.

We now discuss the start and finish rules, which transfer between the polymorphic and monomorphic typing judgments. The start rule, (BQ-SD-START), drops the quantifier and splits the type scheme into its constraint and monotype. Since the variables $\bar{\mathsf{X}}$ are bound, they do not conflict with any free type variable that could be present in the environment. In the HM(X) framework [36], quantifier elimination is of the form

$$(\text{HMX-START})\ \frac{\Gamma \vdash_{\mathsf{p}} e : \forall\bar{\mathsf{X}}|C.\mathsf{T} \qquad \mathsf{true} \Vdash \theta_{\bar{\mathsf{X}}}(C)}{\mathsf{true}; \Gamma \vdash_{\mathsf{m}} e : \theta_{\bar{\mathsf{X}}}(\mathsf{T})}$$

where $\theta_{\bar{\mathsf{X}}}$ is a ground substitution, requiring in essence that the monotype be grounded with a substitution satisfying C. Aside from the non syntax-directed nature of the rule, this formulation would restrict the expressiveness of our type system due to the absence of constraint in the polymorphic typing judgment. For instance, one might derive that under the environment $y : \mathsf{Y}$ the selection $y.l$ has a type $\forall\mathsf{X}|\mathsf{Y} <: \{l : \mathsf{X}\}.\mathsf{X}$. The (HMX-START) rule does not allow to use this expression further since only X is instantiated when checked. In HM(X), one could have an additional constraint such as $\mathsf{Y} <: \{y : \mathsf{Int}\}$ in the polymorphic typing judgment making the type $\forall\mathsf{X}|\mathsf{Y} <: \{l : \mathsf{X}\}.\mathsf{X}$ instantiable. We choose to restrict types appearing in the polymorphic typing judgment to valid types. This restriction makes (HMX-START)'s check unnecessary and allows us to lazily verify the satisfiabiliy of the constraint at the end of the derivation via the (BQ-SD-FIN) rule.

The finish rule, (BQ-SD-FIN), concludes a monomorphic typing derivation and generalizes the inferred type, producing a type scheme. We verify the satisfiability of the synthesized constraint to enforce that only well-typed expressions appear in the polymorphic typing judgment. The essence of this type scheme introduction rule is comparable to that of Aiken and Wimmers [2]. For algorithmic purposes, we specify explicitly the generalized type variables $\bar{\mathsf{X}}$. Sulzmann [36] discusses four flavors of type scheme introduction in constraint-based type inference systems: no satisfiability check, weak satisfiability check, strong satisfiability check, and duplication. In the absence of constraints on polymorphic typing judgments, the distinction between Sulzmann's strong and weak satisfiability checks disappears. Hence our system has both the strong and weak satisfiability checks according to Sulzmann's taxonomy.

3.1.2 Declarative Rules

In addition to the syntax-directed rules, we consider two declarative rules. These are introduced to simplify the understanding of the type system and ease the comparative arguments that follow.

$$\text{(BQ-DEC-ABS)}\ \frac{C;\Gamma\cdot x:\forall\emptyset|\mathsf{true}.\mathsf{T}_1 \vdash_{\mathsf{m}} e:\mathsf{T}_2}{C;\Gamma\vdash_{\mathsf{m}} \lambda x.e:\mathsf{T}_1\rightarrow\mathsf{T}_2}\qquad\qquad \text{(BQ-DEC-SUB)}\ \frac{\Gamma\vdash_{\mathsf{p}} e:\mathsf{S}_1 \qquad \mathsf{S}_1 <: \mathsf{S}_2}{\Gamma\vdash_{\mathsf{p}} e:\mathsf{S}_2}$$

The (BQ-DEC-ABS) is the standard Hindley/Milner style declarative abstraction rule and the (BQ-DEC-SUB) rule is an extended subsumption rule. It is easy to see that the (BQ-SD-ABS) is a narrower version of (BQ-DEC-ABS). This is formalized in the following lemma.

Lemma 3.1.1 *The rule* (BQ-SD-ABS) *is derivable from* (BQ-DEC-ABS).

Proof (lemma 3.1.1) Simply instantiate T_1 to a type variable X.

3.1.3 Type Systems

Based on the typing rules, we define three type systems. The BQ type system consists of the syntax-directed rules and represents the core of the type inference algorithm. The type system BQ^{sub,abs} has a similar flavor as Hindley/Milner's and is used later in our comparison with Hindley/Milner. Finally, the BQ^{abs} system is a mix between BQ and BQ^{sub,abs} whose only purpose is to make the comparative arguments simpler.

Definition 3.1.1 *The type system BQ consists of all the* (BQ-SD) *rules.*

Definition 3.1.2 *The type system BQ^{abs} is the BQ type system where the* (BQ-SD-ABS) *rule is substituted for the* (BQ-DEC-ABS) *rule.*

Definition 3.1.3 *The type system BQ^{sub,abs} extends the BQ^{abs} type system by adding the* (BQ-DEC-SUB) *rule.*

3.2 Properties

In this section, we compare the three type systems introduced previously.

We compare the type systems in pairs and decouple the two comparison directions. We start by showing that BQ is embedded (definition 3.2.1) in BQ^{abs}, which in turn is embedded in BQ^{sub,abs}. Then, we prove depth and width weakening for BQ^{sub,abs}, which, thanks to the embedding property, also applies to BQ and BQ^{abs}.

We continue in the reverse direction, and show that the expressiveness of BQ^{sub,abs} is comparable to the expressiveness of BQ^{abs}, which in turn is comparable to BQ's. These comparisons are done at a semantic level for inductiveness and are rather lengthy. One can summarize the comparison between BQ^{sub,abs} and BQ^{abs} as "if it is derivable in BQ^{sub,abs}, then we can derive a similar or more precise judgment in BQ^{abs}". Similarly, the comparison between BQ^{abs} and BQ can be summarized as "if it is derivable in BQ^{abs}, then we can derive a structurally similar judgment in BQ and highlight a mapping between the two derivations". Based on the pairwise comparisons, we can conclude by showing that BQ derives a more precise type than BQ^{sub,abs} (corollary 3.2.4).

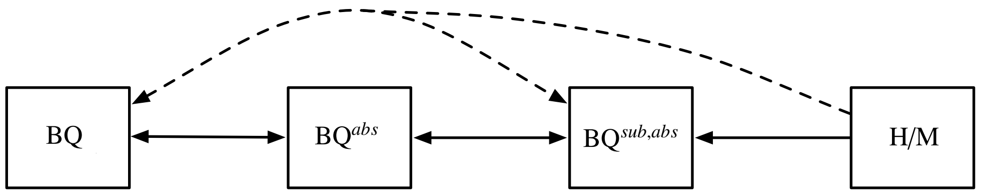

Figure 3.1: Outline of the comparisons of BQ, BQ^{abs}, BQ^{sub,abs} and Hindley/Milner. The dotted comparisons are derived from the solid ones.

Definition 3.2.1 *We say that a type system Ω_1 is embedded in a type system Ω_2, written $\Omega_1 \subseteq \Omega_2$, if all derivations in Ω_1 are derivations in Ω_2.*

Theorem 3.2.1 *The embedding relation is reflexive and transitive.*

Theorem 3.2.2 *$BQ \subseteq BQ^{abs} \subseteq BQ^{sub,abs}$.*

Proof (theorem 3.2.2) The embedding of BQ in BQ^{abs} follows from (lemma 3.1.1), and the embedding of BQ^{abs} in BQ^{sub,abs} is direct since BQ^{sub,abs} extends BQ^{abs}.

Definition 3.2.2 *A derivation $C; \Gamma \vdash_m e : T$ is more precise than a derivation $C'; \Gamma' \vdash_m e : T'$ if $\Gamma' <: \Gamma$ and for all ground assignments ϕ and all ground substitutions $\theta_{\bar{X}}$ there exists a ground substitution $\theta_{\bar{Y}}$ such that $\phi \models \theta_{\bar{X}}(C) \implies \phi \models \theta_{\bar{Y}}(C')$ and $\phi \models \theta_{\bar{Y}}(T') <: \theta_{\bar{X}}(T)$, where $\bar{X} = \mathrm{ftv}(C) \cup \mathrm{ftv}(T) \setminus \mathrm{ftv}(\Gamma)$ and $\bar{Y} = \mathrm{ftv}(C') \cup \mathrm{ftv}(T') \setminus \mathrm{ftv}(\Gamma')$.*

Theorem 3.2.3 (Weakening) *If $\Gamma' <: \Gamma$ and $\Gamma \vdash_p e : S$ in BQ^{sub,abs} then $\Gamma' \vdash_p e : S'$ where $S' <: S$, and if $C; \Gamma \vdash_m e : T$ in BQ^{sub,abs} then a more precise derivation $C'; \Gamma' \vdash_m e : T'$ can be derived.*

Proof (theorem 3.2.3) By induction on the typing rules.

$\boxed{\textsf{(BQ-SD-BASE)}}$ Direct since for any constant c, T_c does not depend on the environment.

$\boxed{\textsf{(BQ-SD-VAR)}}$ Suppose that $\Gamma \vdash_{\mathsf{p}} x : \mathsf{S}$ and $\Gamma' <: \Gamma$. By rule hypothesis $\Gamma(x) = \mathsf{S}$ and by definition $\Gamma'(x) = \mathsf{S}'$ such that $\mathsf{S}' <: \mathsf{S}$. Therefore, by applying the (BQ-SD-VAR) rule we have $\Gamma' \vdash_{\mathsf{p}} x : \mathsf{S}'$.

$\boxed{\textsf{(BQ-SD-START)}}$ Suppose $C; \Gamma \vdash_{\mathsf{m}} e : \mathsf{T}$ and $\Gamma' <: \Gamma$. By rule hypothesis, we have $\Gamma \vdash_{\mathsf{p}} e : \forall \bar{\mathsf{X}} | C.\mathsf{T}$ and by induction

$$\Gamma' \vdash_{\mathsf{p}} e : \forall \bar{\mathsf{Y}} | C'.\mathsf{T}' \quad \text{and} \quad \forall \bar{\mathsf{Y}} | C'.\mathsf{T}' <: \forall \bar{\mathsf{X}} | C.\mathsf{T}$$

and after applying the (BQ-SD-START) rule we have

$$C'; \Gamma' \vdash_{\mathsf{p}} e : \mathsf{T}'$$

and by (theorem 2.8.1), for all ground assignments ϕ and all ground substitutions $\theta_{\bar{\mathsf{X}}}$, there exists $\theta_{\bar{\mathsf{Y}}}$ such that

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \implies \phi \models \theta_{\bar{\mathsf{Y}}}(C') \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T})$$

By (lemma 2.8.6), it is safe to assume that $\bar{\mathsf{X}} \subseteq \mathrm{ftv}(C) \cup \mathrm{ftv}(\mathsf{T})$ and $\bar{\mathsf{Y}} \subseteq \mathrm{ftv}(C') \cup \mathrm{ftv}(\mathsf{T}')$. In addition, $\bar{\mathsf{X}}$ and $\bar{\mathsf{Y}}$ do not conflict with $\mathrm{ftv}(\Gamma)$ and $\mathrm{ftv}(\Gamma')$. Let

$$\begin{aligned} \bar{\mathsf{X}}' &= \mathrm{ftv}(C) \cup \mathrm{ftv}(\mathsf{T}) \setminus \mathrm{ftv}(\Gamma) = \mathrm{ftv}(C) \cup \mathrm{ftv}(\mathsf{T}) \\ \bar{\mathsf{Y}}' &= \mathrm{ftv}(C') \cup \mathrm{ftv}(\mathsf{T}') \setminus \mathrm{ftv}(\Gamma') = \mathrm{ftv}(C') \cup \mathrm{ftv}(\mathsf{T}') \end{aligned}$$

and take a ground assignment ϕ, a ground substitution $\theta_{\bar{\mathsf{X}}'}$ and suppose that $\phi \models \theta_{\bar{\mathsf{X}}'}(C)$. Construct ϕ' such that $\phi'(\mathsf{W}) = \theta_{\bar{\mathsf{X}}'}(\mathsf{W})$ where $\mathsf{W} \in \bar{\mathsf{X}}' \setminus \bar{\mathsf{X}}$ and equal to ϕ otherwise. By construction

$$\phi' \models \theta_{\bar{\mathsf{X}}}(C) \text{ and by implication } \phi' \models \theta_{\bar{\mathsf{Y}}}(C') \text{ and } \phi' \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T}).$$

Construct $\theta_{\bar{\mathsf{Y}}'}$ such that $\theta_{\bar{\mathsf{Y}}}(\mathsf{W}) = \phi'(\mathsf{W})$ where $\mathsf{W} \in \bar{\mathsf{Y}}' \setminus \bar{\mathsf{Y}}$ and equal to $\theta_{\bar{\mathsf{Y}}}$ otherwise. By construction

$$\phi \models \theta_{\bar{\mathsf{Y}}'}(C') \quad \text{and} \quad \phi \models \theta_{\bar{\mathsf{Y}}'}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}'}(\mathsf{T})$$

concluding this case.

$\boxed{\textsf{(BQ-SD-LET)}}$ Suppose $C; \Gamma \vdash_{\mathsf{m}} \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \mathsf{T}$ and $\Gamma' <: \Gamma$. By rule hypothesis

$$\Gamma \vdash_{\mathsf{p}} e_1 : \mathsf{S} \quad \text{and} \quad C; \Gamma \cdot x : \mathsf{S} \vdash_{\mathsf{m}} e_2 : \mathsf{T}$$

and by induction $\Gamma' \vdash_{\mathsf{p}} e_1 : \mathsf{S}'$ where $\mathsf{S}' <: \mathsf{S}$. In addition $\Gamma' \cdot x : \mathsf{S}' <: \Gamma \cdot x : \mathsf{S}$, since $\mathsf{S}' <: \mathsf{S}$ and x is a bound variable that can be freely α-renamed. By induction, we therefore have $C'; \Gamma' \cdot x : \mathsf{S}' \vdash_{\mathsf{m}} e_2 : \mathsf{T}'$ and for all ground assignments ϕ, and all ground substitutions $\theta_{\bar{\mathsf{X}}}$ there exist a ground substitution $\theta_{\bar{\mathsf{Y}}}$ such that

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \implies \phi \models \theta_{\bar{\mathsf{Y}}}(C') \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T})$$

where $\bar{\mathsf{X}} = \mathrm{ftv}(C) \cup \mathrm{ftv}(\mathsf{T}) \setminus \mathrm{ftv}(\Gamma)$ and $\bar{\mathsf{Y}} = \mathrm{ftv}(C') \cup \mathrm{ftv}(\mathsf{T}') \setminus \mathrm{ftv}(\Gamma')$. We conclude this case by applying (BQ-SD-LET), yielding

$$C'; \Gamma' \vdash_{\mathsf{m}} \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \mathsf{T}'.$$

$\boxed{\text{(BQ-SD-REC)}}$ Suppose

$$C_1 \wedge \cdots \wedge C_n; \Gamma \vdash_{\mathsf{m}} \{l_1 = e_1, \ldots, l_n = e_n\} : \{l_1 : \mathsf{T}_1, \ldots, l_n = \mathsf{T}_n\}$$

and $\Gamma' <: \Gamma$. If n is 0 the result is trivial, we therefore assume that it is greater or equal to 1, let i by an index in $[1, n]$. By rule hypothesis we have $C_i; \Gamma \vdash_{\mathsf{m}} e_i : \mathsf{T}_i$, and by induction $C'_i; \Gamma \vdash_{\mathsf{m}} e_i : \mathsf{T}'_i$ such that, for all ground assignnments ϕ and all ground substitutions $\theta_{\bar{\mathsf{X}}_i}$, there exists $\theta_{\bar{\mathsf{Y}}_i}$ such that

$$\phi \models \theta_{\bar{\mathsf{X}}_i}(C_i) \implies \phi \models \theta_{\bar{\mathsf{Y}}_i}(C'_i) \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}_i}(\mathsf{T}'_i) <: \theta_{\bar{\mathsf{X}}_i}(\mathsf{T}_i)$$

where $\bar{\mathsf{X}}_i = \mathrm{ftv}(C_i) \cup \mathrm{ftv}(\mathsf{T}_i) \setminus \mathrm{ftv}(\Gamma)$ and $\bar{\mathsf{Y}}_i = \mathrm{ftv}(C'_i) \cup \mathrm{ftv}(\mathsf{T}'_i) \setminus \mathrm{ftv}(\Gamma')$.

It is important to note that if we take $i \neq j$ then $\bar{\mathsf{X}}_i \# \bar{\mathsf{X}}_j$[1] and $\bar{\mathsf{Y}}_i \# \bar{\mathsf{Y}}_j$. Indeed, variable sharing across separate derivations occurs via the environment.

Let $\bar{\mathsf{X}} = \mathrm{ftv}\,(C_1 \wedge \cdots) \cup \mathrm{ftv}\,(\{l_1 : \mathsf{T}_1, \cdots\}) \setminus \mathrm{ftv}(\Gamma)$ and $\bar{\mathsf{Y}} = \mathrm{ftv}\left(C'_1 \wedge \cdots\right) \cup \mathrm{ftv}\left(\left\{l_1 : \mathsf{T}'_1, \cdots\right\}\right) \setminus \mathrm{ftv}(\Gamma')$. It is clear that $\bar{\mathsf{X}}_i \subseteq \bar{\mathsf{X}}$ and $\bar{\mathsf{Y}}_i \subseteq \bar{\mathsf{Y}}$. Take a ground assignment ϕ and ground substitution $\theta_{\bar{\mathsf{X}}}$ such that

$$\phi \models \theta_{\bar{\mathsf{X}}}\,(C_1 \wedge \cdots \wedge C_n)\,.$$

Since $\mathrm{ftv}(C_i) = \bar{\mathsf{X}}_i \subseteq \bar{\mathsf{X}}$ we have

$$\phi \models \theta_{\bar{\mathsf{X}}_i}\,(C_i)$$

and by implication

$$\phi \models \theta_{\bar{\mathsf{Y}}_i}(C'_i) \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}_i}(\mathsf{T}'_i) <: \theta_{\bar{\mathsf{X}}_i}(\mathsf{T}_i)$$

We can merge all ground substitutions into $\theta_{\bar{\mathsf{Y}}}$ which allows us to conclude

$$\phi \models \theta_{\bar{\mathsf{Y}}}(C_1 \wedge \cdots \wedge C_n) \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}'_i) <: \theta_{\bar{\mathsf{X}}}(\mathsf{T}_i)$$

and by (BQ-S-REC) and (BQ-C-SUB)

$$\phi \models \theta_{\bar{\mathsf{Y}}}(\{l_1 : \mathsf{T}'_1, \cdots, l_n : \mathsf{T}'_n\}) <: \theta_{\bar{\mathsf{X}}}(\{l_1 : \mathsf{T}_1, \cdots, l_n : \mathsf{T}_n\})$$

[1]That is $\bar{\mathsf{X}}_i \cap \bar{\mathsf{X}}_j = \emptyset$.

concluding this case.

$\boxed{\text{(BQ-SD-SEL)}}$ Suppose $C \wedge \mathsf{T} <: \{l : \mathsf{W}\}\,; \Gamma \vdash_{\mathsf{m}} e.l : \mathsf{W}$ and $\Gamma' <: \Gamma$. By rule hypothesis

$$C; \Gamma \vdash_{\mathsf{m}} e : \mathsf{T} \quad \text{and} \quad \mathsf{W} \text{ is fresh.}$$

By induction $C'; \Gamma' \vdash_{\mathsf{m}} e : \mathsf{T}'$ and for all ground assignments ϕ and all ground substitutions $\theta_{\bar{\mathsf{X}}}$, there exist a ground substitution $\theta_{\bar{\mathsf{Y}}}$ such that

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \implies \phi \models \theta_{\bar{\mathsf{Y}}}(C') \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T})$$

where $\bar{\mathsf{X}} = \text{ftv}(C) \cup \text{ftv}(\mathsf{T}) \setminus \text{ftv}(\Gamma)$ and $\bar{\mathsf{Y}} = \text{ftv}(C') \cup \text{ftv}(\mathsf{T}') \setminus \text{ftv}(\Gamma')$. Applying the (BQ-SD-SEL) rule yields

$$C' \wedge \mathsf{T}' <: \{l : \mathsf{W}\}\,; \Gamma' \vdash_{\mathsf{m}} e.l : \mathsf{W}$$

since W is fresh. Let

$$\begin{aligned} \bar{\mathsf{X}}' &= \text{ftv}\,(C \wedge \mathsf{T} <: \{l : \mathsf{W}\}) \cup \text{ftv}(\mathsf{T}) \setminus \text{ftv}(\Gamma) = \bar{\mathsf{X}} \cup \mathsf{W} \\ \bar{\mathsf{Y}}' &= \text{ftv}\,(C' \wedge \mathsf{T}' <: \{l : \mathsf{W}\}) \cup \text{ftv}(\mathsf{T}') \setminus \text{ftv}(\Gamma') = \bar{\mathsf{Y}} \cup \mathsf{W} \end{aligned}$$

and take an arbitrary ground substitution $\theta_{\bar{\mathsf{X}}'}$ and a ground assignment ϕ satisfying $\theta_{\bar{\mathsf{X}}'}(C \wedge \mathsf{T} <: \{l : \mathsf{W}\})$. Take the ground assignment ϕ' similar to ϕ in every position except for $\phi'(\mathsf{W}) = \theta_{\bar{\mathsf{X}}'}(\mathsf{W})$. By construction, $\phi' \models \theta_{\bar{\mathsf{X}}}(C)$ and by implication $\phi' \models \theta_{\bar{\mathsf{Y}}}(C')$ and $\phi' \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T})$, for some ground substitution $\theta_{\bar{\mathsf{Y}}}$. Since W is fresh, it does not appear in C, C', T nor in T'. Therefore

$$\phi \models \theta_{\bar{\mathsf{Y}}'}(C') \quad \text{and} \quad \phi \models \theta_{\bar{\mathsf{Y}}'}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}'}(\mathsf{T})$$

where we select $\theta_{\bar{\mathsf{Y}}'}$ such that $\theta_{\bar{\mathsf{Y}}'}(\mathsf{W}) = \theta_{\bar{\mathsf{X}}'}(\mathsf{W})$.

By interpretation of $\phi \models \theta_{\bar{\mathsf{X}}'}(C \wedge \mathsf{T} <: \{l : \mathsf{W}\})$

$$\phi \models \theta_{\bar{\mathsf{X}}'}(C) \quad \text{and} \quad \phi \models \theta_{\bar{\mathsf{X}}'}(\mathsf{T} <: \{l : \mathsf{W}\}).$$

Pushing the ground substitution we have $\phi \models \theta_{\bar{\mathsf{X}}'}(\mathsf{T}) <: \theta_{\bar{\mathsf{X}}'}(\{l : \mathsf{W}\})$. As $\theta_{\bar{\mathsf{X}}'}(\mathsf{W}) = \theta_{\bar{\mathsf{Y}}'}(\mathsf{W})$ and $\phi \models \theta_{\bar{\mathsf{Y}}'}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}'}(\mathsf{T})$ we can derive using (BQ-C-AND)

$$\phi \models \theta_{\bar{\mathsf{Y}}'}(C' \wedge \mathsf{T}'\, \{l : \mathsf{W}\}).$$

Finally, $\phi \models \theta_{\bar{\mathsf{Y}}'}(\mathsf{W}) <: \theta_{\bar{\mathsf{X}}'}(\mathsf{W})$ since $\theta_{\bar{\mathsf{Y}}'}(\mathsf{W}) = \theta_{\bar{\mathsf{X}}'}(\mathsf{W})$, concluding this case.

$\boxed{\text{(BQ-DEC-ABS)}}$ Suppose $C; \Gamma \vdash_{\mathsf{m}} \lambda x.e : \mathsf{T}_1 \to \mathsf{T}_2$ and $\Gamma' <: \Gamma$. By rule hypothesis

$$C; \Gamma \cdot x : \forall \emptyset | \mathsf{true}.\mathsf{T}_1 \vdash_{\mathsf{m}} e : \mathsf{T}_2$$

and $\Gamma' \cdot x : \forall\emptyset|\mathsf{true}.\mathsf{T}_1 <: \Gamma \cdot x : \forall\emptyset|\mathsf{true}.\mathsf{T}_1$ as x is bound in e and therefore does not conflict with Γ'. By induction

$$C'; \Gamma' \cdot x : \forall\emptyset|\mathsf{true}.\mathsf{T}_1 \vdash_{\mathsf{m}} e : \mathsf{T}_2' \tag{3.1}$$

and for all ground assignments ϕ and all ground substitutions $\theta_{\bar{\mathsf{X}}}$, there exists a ground substitution $\theta_{\bar{\mathsf{Y}}}$ such that

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \implies \phi \models \theta_{\bar{\mathsf{Y}}}(C') \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}_2') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T}_2)$$

where

$$\begin{aligned}\bar{\mathsf{X}} &= \mathrm{ftv}(C) \cup \mathrm{ftv}(\mathsf{T}_2) \setminus \mathrm{ftv}\,(\Gamma \cdot x : \forall\emptyset|\mathsf{true}.\mathsf{T}_1) = \mathrm{ftv}(C) \cup \mathrm{ftv}(\mathsf{T}_2) \setminus \mathrm{ftv}(\Gamma) \cup \mathrm{ftv}(\mathsf{T}_1)\\ \bar{\mathsf{Y}} &= \mathrm{ftv}(C') \cup \mathrm{ftv}(\mathsf{T}_2') \setminus \mathrm{ftv}\,(\Gamma \cdot x : \forall\emptyset|\mathsf{true}.\mathsf{T}_1) = \mathrm{ftv}(C') \cup \mathrm{ftv}(\mathsf{T}_2') \setminus \mathrm{ftv}(\Gamma') \cup \mathrm{ftv}(\mathsf{T}_1)\end{aligned}$$

Applying (BQ-SD-ABS) to (3.1) yields

$$C'; \Gamma' \vdash_{\mathsf{m}} \lambda x.e : \mathsf{T}_1 \to \mathsf{T}_2'.$$

Let

$$\begin{aligned}\bar{\mathsf{X}}' &= \mathrm{ftv}\,(C) \cup \mathrm{ftv}(\mathsf{T}_1 \to \mathsf{T}_2) \setminus \mathrm{ftv}(\Gamma) = \bar{\mathsf{X}} \cup \mathrm{ftv}(\mathsf{T}_1)\\ \bar{\mathsf{Y}}' &= \mathrm{ftv}\,(C') \cup \mathrm{ftv}(\mathsf{T}_1 \to \mathsf{T}_2') \setminus \mathrm{ftv}(\Gamma') = \bar{\mathsf{Y}} \cup \mathrm{ftv}(\mathsf{T}_1)\end{aligned}$$

Let $\mathsf{W} \in \mathrm{ftv}(\mathsf{T}_1)$ and take two arbitrary ground substitution $\theta_{\bar{\mathsf{X}}'}$, $\theta_{\bar{\mathsf{Y}}'}$ such that $\theta_{\bar{\mathsf{X}}'}(\mathsf{W}) = \theta_{\bar{\mathsf{Y}}'}(\mathsf{W})$ and a ground assignment ϕ satisfying $\theta_{\bar{\mathsf{X}}'}(C)$. Take ϕ' equal to ϕ except for $\phi'(\mathsf{W}) = \phi\,(\theta_{\bar{\mathsf{X}}'}(\mathsf{W}))$ and $\phi'(\mathsf{W}) = \phi\,(\theta_{\bar{\mathsf{Y}}'}(\mathsf{W}))$. By construction, ϕ' satisfies $\theta_{\bar{\mathsf{X}}}(C)$, and by implication

$$\phi' \models \theta_{\bar{\mathsf{Y}}}(C') \quad \text{and} \quad \phi' \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}_2') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T}_2)$$

thus $\phi \models \theta_{\bar{\mathsf{Y}}'}(C')$ and $\phi \models \theta_{\bar{\mathsf{Y}}'}(\mathsf{T}_1 \to \mathsf{T}_2') <: \theta_{\bar{\mathsf{X}}'}(\mathsf{T}_1 \to \mathsf{T}_2)$ concluding this case.

$\boxed{\text{(BQ-SD-APP)}}$ Suppose

$$\underbrace{C_1 \wedge C_2 \wedge \mathsf{T}_1 <: \mathsf{X} \to \mathsf{Y} \wedge \mathsf{T}_2 <: \mathsf{X}}_{C}; \Gamma \vdash_{\mathsf{m}} e_1\ e_2 : \mathsf{Y}$$

and $\Gamma' <: \Gamma$. By rule hypothesis

$$C_1; \Gamma \vdash_{\mathsf{m}} e_1 : \mathsf{T}_1 \quad \text{and} \quad C_2; \Gamma \vdash_{\mathsf{m}} e_2 : \mathsf{T}_2 \quad \text{and } \mathsf{X}, \mathsf{Y} \text{ are fresh.}$$

By induction

$$C_1'; \Gamma' \vdash_{\mathsf{m}} e_1 : \mathsf{T}_1' \quad \text{and} \quad C_2'; \Gamma' \vdash_{\mathsf{m}} e_2 : \mathsf{T}_2'$$

and for all ground assignments ϕ and all ground substitutions $\theta_{\bar{\mathsf{X}}_1}, \theta_{\bar{\mathsf{X}}_2}$, there exists ground substitutions $\theta_{\bar{\mathsf{Y}}_1}, \theta_{\bar{\mathsf{Y}}_2}$ such that

$$\begin{aligned}\phi \models \theta_{\bar{\mathsf{X}}_1}(C_1) &\implies \phi \models \theta_{\bar{\mathsf{Y}}_1}(C_1') \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}_1}(\mathsf{T}_1') <: \theta_{\bar{\mathsf{X}}_1}(\mathsf{T}_1)\\ \phi \models \theta_{\bar{\mathsf{X}}_2}(C_2) &\implies \phi \models \theta_{\bar{\mathsf{Y}}_2}(C_2') \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}_2}(\mathsf{T}_2') <: \theta_{\bar{\mathsf{X}}_2}(\mathsf{T}_2)\end{aligned}$$

where $\bar{\mathsf{X}}_1 = \mathrm{ftv}(C_1) \cup \mathrm{ftv}(\mathsf{T}_1) \setminus \mathrm{ftv}(\Gamma)$, $\bar{\mathsf{X}}_2 = \mathrm{ftv}(C_2) \cup \mathrm{ftv}(\mathsf{T}_2) \setminus \mathrm{ftv}(\Gamma)$, $\bar{\mathsf{Y}}_1 = \mathrm{ftv}(C_1') \cup \mathrm{ftv}(\mathsf{T}_1') \setminus \mathrm{ftv}(\Gamma')$, $\bar{\mathsf{Y}}_2 =$

$\text{ftv}(C'_2) \cup \text{ftv}(\mathsf{T}'_2) \setminus \text{ftv}(\Gamma')$. We have $\bar{\mathsf{X}}_1 \# \bar{\mathsf{X}}_2$ and $\bar{\mathsf{Y}}_1 \# \bar{\mathsf{Y}}_2$ as explained in the (BQ-SD-REC) case. Applying the (BQ-SD-APP) gives

$$\underbrace{C'_1 \wedge C'_2 \wedge \mathsf{T}'_1 <: \mathsf{X} \to \mathsf{Y} \wedge \mathsf{T}'_2 <: \mathsf{X}}_{C'}; \Gamma' \vdash_{\mathsf{m}} e_1\, e_2 : \mathsf{Y}$$

Let

$$\bar{\mathsf{X}} = \text{ftv}(C) \cup \text{ftv}(\mathsf{Y}) \setminus \text{ftv}(\Gamma) = \bar{\mathsf{X}}_1 \cup \bar{\mathsf{X}}_2 \cup \{\mathsf{X}, \mathsf{Y}\}$$
$$\bar{\mathsf{Y}} = \text{ftv}(C') \cup \text{ftv}(\mathsf{Y}) \setminus \text{ftv}(\Gamma') = \bar{\mathsf{Y}}_1 \cup \bar{\mathsf{Y}}_2 \cup \{\mathsf{X}, \mathsf{Y}\}$$

Take a ground assignment ϕ and a ground substitution $\theta_{\bar{\mathsf{X}}}$ such that $\theta_{\bar{\mathsf{X}}}(C)$ is satisfied. By interpetation

$$\phi \models \theta_{\bar{\mathsf{X}}}(C_1) \quad \phi \models \theta_{\bar{\mathsf{X}}}(C_2) \quad \phi \models \theta_{\bar{\mathsf{X}}}(\mathsf{T}_1 <: \mathsf{X} \to \mathsf{Y}) \quad \phi \models \theta_{\bar{\mathsf{X}}}(\mathsf{T}_2 <: \mathsf{X}).$$

We can restrict the domain of $\theta_{\bar{\mathsf{X}}}$ to $\theta_{\bar{\mathsf{X}}_1}$ and $\theta_{\bar{\mathsf{X}}_2}$ and we have

$$\phi \models \theta_{\bar{\mathsf{X}}_1}(C_1) \quad \phi \models \theta_{\bar{\mathsf{X}}_2}(C_2)$$

and by implication

$$\phi \models \theta_{\bar{\mathsf{Y}}_1}(C'_1) \quad \phi \models \theta_{\bar{\mathsf{Y}}_1}(\mathsf{T}'_1) <: \theta_{\bar{\mathsf{X}}_1}(\mathsf{T}_1) \quad \phi \models \theta_{\bar{\mathsf{Y}}_2}(C'_2) \quad \phi \models \theta_{\bar{\mathsf{Y}}_1}(\mathsf{T}'_2) <: \theta_{\bar{\mathsf{X}}_1}(\mathsf{T}_2)$$

Construct $\theta_{\bar{\mathsf{Y}}}$ as the union of $\theta_{\bar{\mathsf{Y}}_1}$ and $\theta_{\bar{\mathsf{Y}}_2}$ such that $\theta_{\bar{\mathsf{Y}}}(\mathsf{X}) = \theta_{\bar{\mathsf{X}}}(\mathsf{X})$ and $\theta_{\bar{\mathsf{Y}}}(\mathsf{Y}) = \theta_{\bar{\mathsf{X}}}(\mathsf{Y})$. We have

$$\phi \models \theta_{\bar{\mathsf{Y}}}(C'_1) \quad \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}'_1 <: \mathsf{X} \to \mathsf{Y}) \quad \phi \models \theta_{\bar{\mathsf{Y}}}(C'_2) \quad \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}'_2 <: \mathsf{Y})$$

which concludes this case.

$\boxed{\text{(BQ-SD-FIN)}}$ Suppose $\Gamma \vdash_{\mathsf{p}} e : \forall \bar{\mathsf{X}} | C.\mathsf{T}$ and $\Gamma' <: \Gamma$. By rule hypothesis

$$C; \Gamma \vdash_{\mathsf{m}} e : \mathsf{T} \quad \bar{\mathsf{X}} = \text{ftv}(\mathsf{T}) \cup \text{ftv}(C) \setminus \text{ftv}(\Gamma) \quad \models C.$$

By induction $C'; \Gamma' \vdash_{\mathsf{m}} e : \mathsf{T}'$ and for all ground assignments ϕ

$$\phi \models \theta_{\bar{\mathsf{X}}}(C) \implies \phi \models \theta_{\bar{\mathsf{Y}}}(C') \text{ and } \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T})$$

where $\bar{\mathsf{Y}} = \text{ftv}(\mathsf{T}') \cup \text{ftv}(C') \setminus \text{ftv}(\Gamma')$. Since C is satisfiable, there exists a ground assignment ϕ and a ground substitution $\theta_{\text{ftv}(C)}$ such that $\phi \models \theta_{\text{ftv}(C)}(C)$. From this, one can construct a ground assignment ϕ' and a ground substitution $\theta_{\bar{\mathsf{X}}}$ such that $\phi(\theta_{\text{ftv}(C)}(C)) = \phi'(\theta_{\bar{\mathsf{X}}}(C))$ and therefore $\phi' \models \theta_{\bar{\mathsf{X}}}(C)$. By implication $\phi' \models \theta_{\bar{\mathsf{Y}}}(C')$ and doing the same 'backward conversion' shows that C' is satisfiable. By applying the (BQ-SD-FIN) rule we get

$$\Gamma' \vdash_{\mathsf{p}} e : \forall \bar{\mathsf{Y}} | C'.\mathsf{T}'$$

and we are going to show that $\forall \bar{\mathsf{Y}} | C'.\mathsf{T}' <: \forall \bar{\mathsf{X}} | C.\mathsf{T}$. Fix a ground assignment ϕ and let $\mathsf{T}^\star \in [\![\forall \bar{\mathsf{X}} | C.\mathsf{T}]\!]_\phi$. By definition of (definition 2.6.1) and (definition 2.8.1)

$$\phi\,(\theta_{\bar{\mathsf{X}}}(\mathsf{T})) <: \mathsf{T}^\star \quad \text{and} \quad \phi \models \theta_{\bar{\mathsf{X}}}(C)$$

by implication

$$\phi \models \theta_{\bar{\mathsf{Y}}}(C') \quad \text{and} \quad \phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \theta_{\bar{\mathsf{X}}}(\mathsf{T})$$

for some ground substitution $\theta_{\bar{\mathsf{Y}}}$. By transitivity of the subtyping relation (theorem 2.6.2)

$$\phi \models \theta_{\bar{\mathsf{Y}}}(\mathsf{T}') <: \mathsf{T}^\star$$

which is equivalent to $\phi\,(\theta_{\bar{\mathsf{Y}}}(\mathsf{T}')) <: \phi\,(\mathsf{T}^\star)$, and $\phi\,(\mathsf{T}^\star) = \mathsf{T}^\star$ since $\mathsf{T}^\star$ is a ground type. Therefore

$$\mathsf{T}^\star \in [\![\forall \bar{\mathsf{Y}} | C'.\mathsf{T}']\!]_\phi$$

proving that $[\![\forall \bar{\mathsf{X}} | C.\mathsf{T}]\!]_\phi \subseteq [\![\forall \bar{\mathsf{Y}} | C'.\mathsf{T}']\!]_\phi$. By definition (definition 2.8.3) $\forall \bar{\mathsf{Y}} | C'.\mathsf{T}' <: \forall \bar{\mathsf{X}} | C.\mathsf{T}$, concluding this case.

$\boxed{\text{(BQ-DEC-SUB)}}$ Immediate.

Corollary 3.2.1 *The weakening theorem applies to BQ and BQ^{abs}.*

Definition 3.2.3 *We define an expression's context as an expression with a single hole*

$$E ::= [\cdot] \mid \lambda x.E \mid E_1\ e_2 \mid e_1\ E_2 \mid \textit{let}\ x = E_1\ \textit{in}\ e_2 \mid \textit{let}\ x = e_1\ \textit{in}\ E_2 \mid \{l_1 : e_1, \ldots, l_k : E_k, \ldots, l_n : e_n\} \mid E.l.$$

Theorem 3.2.4 *Let $\Gamma \vdash_p e : \mathsf{S}_1$, if $\Gamma \vdash_p E[e] : \mathsf{S}_2$ then $\Gamma \cdot x : \mathsf{S}_1 \vdash_p E[x] : \mathsf{S}_2$, and similarly if $\Gamma \vdash_p E[e] : T_2$ then $\Gamma \cdot x : \mathsf{S}_1 \vdash_p E[x] : T_2$ with $x \notin \text{dom}(\Gamma)$ and is not bound in E.*

Proof (theorem 3.2.4) Direct since the typing rules do not inspect an expression's shape once the expression has been typed.

Corollary 3.2.2 *If $\Gamma \vdash_p e : \mathsf{S}_1$, $\Gamma \vdash_p E[e] : \mathsf{S}_2$ and $\Gamma \vdash_p e' : \mathsf{S}'_1$ with $\mathsf{S}'_1 <: \mathsf{S}_1$ then $\Gamma \vdash_p E[e'] : \mathsf{S}'_2$ with $\mathsf{S}'_2 <: \mathsf{S}_2$.*

If $C_1; \Gamma \vdash_p e : T_1$, $C_2; \Gamma \vdash_p E[e] : T_2$ and a more precise derivation $C'_1\Gamma \vdash_p e' : T'_1$ then $C'_2; \Gamma \vdash_p E[e'] : T'_2$ is a more precise derivation.

Proof (corollary 3.2.2) Follows from (theorem 3.2.3) and (theorem 3.2.4).

Theorem 3.2.5 *If $\Gamma \vdash_p e : \mathsf{S}$ is derivable in BQ^{sub,abs} then $\Gamma \vdash_p e : \mathsf{S}'$ is derivable in BQ^{abs} with $\mathsf{S}' <: \mathsf{S}$.*

Proof (theorem 3.2.5) By full induction on the number of uses of the (BQ-DEC-SUB) rule in a derivation.

We suppose that if there are n or fewer uses of the (BQ-DEC-SUB) rule we have a derivation in BQ^{abs} deriving S' such that $\mathsf{S}' <: \mathsf{S}$. Consider a derivation with $n+1$ uses of the (BQ-DEC-SUB) rule. Pick one use of the (BQ-DEC-SUB) rule dividing the derivation into two sub-derivations Δ_1 and Δ_2 deriving S_1, S_2 in which the (BQ-DEC-SUB) rule is used n times of fewer. Inductively, we have two sub-derivations Δ'_1 and Δ'_2 in BQ^{abs} deriving S'_1, S'_2 with $\mathsf{S}'_1 <: \mathsf{S}_1$ and $\mathsf{S}'_2 <: \mathsf{S}_2$. If Δ'_2 is not an empty derivation (in which case we can conclude immediately), the first rule is either (BQ-SD-START) or (BQ-SD-LET). We can conclude using (corollary 3.2.2) for the first case and weakening (theorem 3.2.3) for the second.

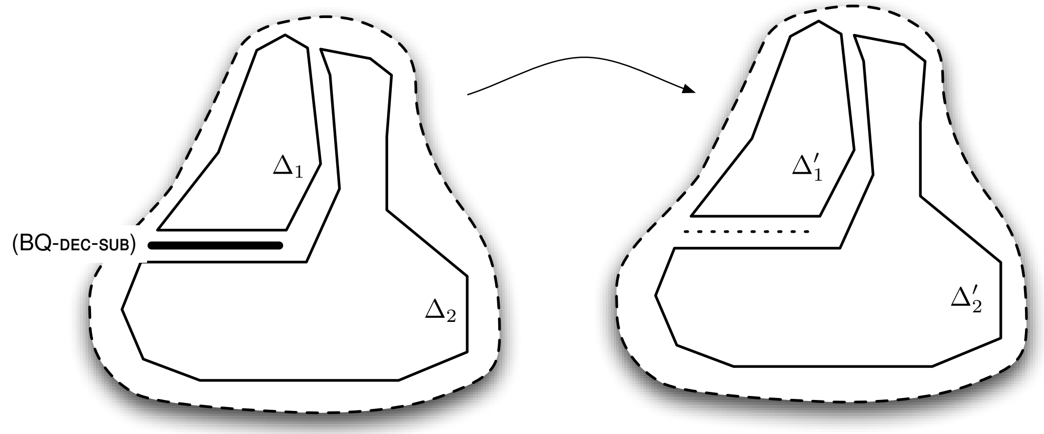

Figure 3.2: Comparison of derivations in BQ^{abs} and BQ^{abs}.

Theorem 3.2.6 *If $\Gamma \vdash_p e : \forall\bar{\mathsf{X}}|C.\mathsf{T}$ is derivable in BQ^{abs} then for all maps ψ such that $\psi(\Gamma') = \Gamma$, $\Gamma' \vdash_p e : \forall\bar{\mathsf{Y}}|C'.\mathsf{T}'$ is derivable and for all substitutions $\theta_{\bar{\mathsf{X}}}$ there exists $\theta_{\bar{\mathsf{Y}}}$ with $\psi(\theta_{\bar{\mathsf{Y}}}(C')) = \theta_{\bar{\mathsf{X}}}(C)$, $\psi(\theta_{\bar{\mathsf{Y}}}(\mathsf{T}')) = \theta_{\bar{\mathsf{X}}}(\mathsf{T})$.*

In addition, if $C; \Gamma \vdash_m e : \mathsf{T}$ is derivable in BQ^{abs} then for all maps ψ such that $\psi(\Gamma') = \Gamma$, the derivation $C'; \Gamma' \vdash_m e : \mathsf{T}'$ is derivable with $\psi(C') = C$ and $\psi(\mathsf{T}') = \mathsf{T}$.

Proof (theorem 3.2.6) By induction on the typing rules.

$\boxed{\text{(BQ-SD-BASE)}}$ Immediate since a constant's type does not depend on the environment.

$\boxed{\text{(BQ-SD-VAR)}}$ Suppose $\Gamma \vdash_p x : \mathsf{S}$ and take ψ and Γ' such that $\psi(\Gamma') = \Gamma$. By rule hypothesis $\Gamma(x) = \mathsf{S}$ and therefore $\psi(\Gamma')(x) = \mathsf{S}$ which allows us to conclude this case.

$\boxed{\text{(BQ-SD-START)}}$ For the purpose of the proof, we highlight the silent substitution implied by the (BQ-SD-START) rule yielding

$$\text{(BQ-SD-START)} \quad \frac{\Gamma \vdash_p e : \forall\bar{\mathsf{X}}|C.\mathsf{T}}{\theta_{\bar{\mathsf{X}}}(C); \Gamma \vdash_m e : \theta_{\bar{\mathsf{X}}}(\mathsf{T})}.$$

Suppose $\theta_{\bar{\mathsf{X}}}(C); \Gamma \vdash_m e : \theta_{\bar{\mathsf{X}}}(\mathsf{T})$ and take ψ and Γ' such that $\psi(\Gamma') = \Gamma$. By rule hypothesis $\Gamma \vdash_p e : \forall\bar{\mathsf{X}}|C.\mathsf{T}$ and by induction $\Gamma' \vdash_p e : \forall\bar{\mathsf{Y}}|C'.\mathsf{T}'$ and for all substitutions $\theta_{\bar{\mathsf{X}}}$ there exists $\theta_{\bar{\mathsf{Y}}}$ such that $\psi(\theta_{\bar{\mathsf{Y}}}(C')) = \theta_{\bar{\mathsf{X}}}(C)$ and

$\psi(\theta_{\bar{\mathsf{Y}}}(\mathsf{T}')) = \theta_{\bar{\mathsf{X}}}(\mathsf{T})$. We can therefore apply the (BQ-SD-START) rule and conclude.

$\boxed{\text{(BQ-SD-LET), (BQ-SD-REC), (BQ-SD-SEL), (BQ-DEC-ABS), (BQ-SD-APP)}}$ Immediate.

$\boxed{\text{(BQ-SD-FIN)}}$ Suppose $\Gamma \vdash_{\mathsf{p}} e : \forall\bar{\mathsf{X}}|C.\mathsf{T}$ and take ψ and Γ' such that $\psi(\Gamma') = \Gamma$. By rule hypothesis $C;\Gamma \vdash_{\mathsf{m}} e : \mathsf{T}$, $\bar{\mathsf{X}} = \mathrm{ftv}(\mathsf{T}) \cup \mathrm{ftv}(C) \setminus \mathrm{ftv}(\Gamma)$ and $\models C$. By induction $C';\Gamma' \vdash_{\mathsf{p}} e : \mathsf{T}'$ is derivable with $\psi(C') = C$ and $\psi(\mathsf{T}') = \mathsf{T}$. Since C is satifiable, C' is also satisfiable (there exists a ground assignment ϕ such that $\phi \models C$ thus $\phi \circ \psi \models C'$). Now, let $\bar{\mathsf{Y}} = \mathrm{ftv}(\mathsf{T}') \cup \mathrm{ftv}(C') \setminus \mathrm{ftv}(\Gamma')$ and take a substitution $\theta_{\bar{\mathsf{X}}}$. We are going to show that there exists a substitution $\theta_{\bar{\mathsf{Y}}}$ satisfying $\psi(\theta_{\bar{\mathsf{Y}}}(C')) = \theta_{\bar{\mathsf{X}}}(C)$ and $\psi(\theta_{\bar{\mathsf{Y}}}(\mathsf{T}')) = \theta_{\bar{\mathsf{X}}}(\mathsf{T})$. Indeed $\bar{\mathsf{X}} = \mathrm{ftv}\left(\psi(\bar{\mathsf{Y}})\right)$ as derived below

$$\begin{aligned}
\bar{\mathsf{X}} &= \mathrm{ftv}(\mathsf{T}) \cup \mathrm{ftv}(C) \setminus \mathrm{ftv}(\Gamma) \\
&= \mathrm{ftv}(\psi(\mathsf{T}')) \cup \mathrm{ftv}(\psi(C')) \setminus \mathrm{ftv}(\psi(\Gamma')) \\
&= \mathrm{ftv}(\psi(\mathrm{ftv}(\mathsf{T}'))) \cup \mathrm{ftv}(\psi(\mathrm{ftv}(C'))) \setminus \mathrm{ftv}(\psi(\mathrm{ftv}(\Gamma'))) \\
&= \mathrm{ftv}\,(\psi\,(\mathrm{ftv}(\mathsf{T}') \cup \mathrm{ftv}(C') \setminus \mathrm{ftv}(\Gamma'))) \\
&= \mathrm{ftv}\left(\psi(\bar{\mathsf{Y}})\right).
\end{aligned}$$

We can conclude by using the (BQ-SD-FIN) rule.

Corollary 3.2.3 *If $\Gamma \vdash_p e : \forall\bar{X}|C.T$ is derivable in BQ^{abs} then there exists a map ψ such that $\Gamma' \vdash_p e : \forall\bar{Y}|C'.T'$ is derivable in BQ and for all substitutions $\theta_{\bar{X}}$ there exists $\theta_{\bar{Y}}$ with $\psi(\theta_{\bar{Y}}(C')) = \theta_{\bar{X}}(C)$, $\psi(\theta_{\bar{Y}}(T')) = \theta_{\bar{X}}(T)$.*

In addition, if $C;\Gamma \vdash_m e : T$ is derivable in BQ^{abs} then there exists a map ψ such that $C';\Gamma' \vdash_m e : T'$ is derivable in BQ with $\psi(\Gamma') = \Gamma$, $\psi(C') = C$ and $\psi(T') = T$.

Proof (corollary 3.2.3) Follows from (theorem 3.2.6). When using the (BQ-SD-ABS) rule one can select an environment with a type variable for the BQ derivation and map it to the type chosen in the BQ^{abs} derivation.

Corollary 3.2.4 *If $\Gamma \vdash_p e : S$ is derivable in BQ^{sub,abs} then $\Gamma \vdash_p e : S'$ is derivable in BQ with $S' <: S$.*

Proof (corollary 3.2.4) Follows from (corollary 3.2.3) and (theorem 3.2.5).

Chapter 4

Conservative Extension of Hindley/Milner

In this section, we are going to show that our type system is a conservative extension of Hindley/Milner's type system. For this, we compare Hindley/Milner to BQ^{sub,abs} and show that Hindley/Milner's derivation can be matched.

4.1 Survey of Hindley/Milner

Expressions in Hindley/Milner are generated by the following grammar.

$$e ::= c \mid x \mid \lambda x. \mid e_1\ e_2 \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2.$$

We have constants c (which we suppose are the same as in the BQ family), variables, anonymous functions and let polymorphism. Monomorphic types consist of base types, type variables and functional types.

$$\mathsf{T} ::= \mathsf{B} \mid \mathsf{X} \mid \mathsf{T}_1 \rightarrow \mathsf{T}_2.$$

We further assume that the set of type variables is the same as in the BQ family. One can observe that the expressions and monomorphic types in the BQ family extend those of Hindley/Milner, which can be formalized by the following theorem.

Theorem 4.1.1 *Let e be an expression in Hindley/Milner, then e is an expression in the BQ family. Similarly, if* T *is a monomorphic Hindley/Milner type, then it is also a type in the BQ family.*

Proof (theorem 4.1.1) Follows directly from the definition of expressions and monomorphic types.

Hindley/Milner polymorphic types are either a monomorphic type or a generalized type.

$$\mathsf{S} ::= \mathsf{T} \mid \forall \bar{\mathsf{X}}.\mathsf{T}.$$

In the BQ family, polymorphic types are separate from monomorphic types and are constrained. We thus convert Hindley/Milner polymorphic types to the BQ family.

Definition 4.1.1 *Let* $[\![\cdot]\!]_{H/M \triangleright BQ}$ *denote a type conversion function from Hindley/Milner to the BQ family defined by*

$$[\![T]\!]_{H/M \triangleright BQ} = \forall\emptyset | true.T \qquad\qquad [\![\forall\bar{X}.T]\!]_{H/M \triangleright BQ} = \forall\bar{X} | true.T$$

and we lift this definition to environments

$$[\![\Gamma]\!]_{H/M \triangleright BQ} = \{x : [\![S]\!]_{H/M \triangleright BQ} \mid x : S \in \Gamma\}.$$

Lemma 4.1.1 *Let* S *be a Hindley/Milner polymorphic type, then* $\mathrm{ftv}(S) = \mathrm{ftv}([\![S]\!]_{H/M \triangleright BQ})$.

Proof (lemma 4.1.1) Immediate.

4.1.1 Hindley/Milner Typing Rules

We repeat for clarity Hindley/Milner's type inference. The rules we consider are adapted from Pottier and Rémy [29] but the original presentation of Damas and Milner [7] is very close.

$$\text{(H/M-BASE)}\ \frac{\mathcal{B}(c) = \mathsf{T}_c}{\Gamma \vdash_{H/M} c : \mathsf{T}_c} \qquad \text{(H/M-VAR)}\ \frac{\Gamma(x) = \mathsf{S}}{\Gamma \vdash_{H/M} x : \mathsf{S}} \qquad \text{(H/M-ABS)}\ \frac{\Gamma \cdot x : \mathsf{T}_1 \vdash_{H/M} e : \mathsf{T}_2}{\Gamma \vdash_{H/M} \lambda x.e : \mathsf{T}_1 \to \mathsf{T}_2}$$

$$\text{(H/M-APP)}\ \frac{\Gamma \vdash_{H/M} e_1 : \mathsf{T}_1 \to \mathsf{T}_2 \qquad \Gamma \vdash_{H/M} e_2 : \mathsf{T}_1}{\Gamma \vdash_{H/M} e_1\ e_2 : \mathsf{T}_2} \qquad \text{(H/M-LET)}\ \frac{\Gamma \vdash_{H/M} e_1 : \mathsf{S} \qquad \Gamma \cdot x : \mathsf{S} \vdash_{H/M} e_2 : \mathsf{T}}{\Gamma \vdash_{H/M} \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \mathsf{T}}$$

$$\text{(H/M-GEN)}\ \frac{\Gamma \vdash_{H/M} e : \mathsf{T} \qquad \bar{\mathsf{X}} \# \mathrm{ftv}(\Gamma)}{\Gamma \vdash_{H/M} e : \forall\bar{\mathsf{X}}.\mathsf{T}} \qquad \text{(H/M-INST)}\ \frac{\Gamma \vdash_{H/M} e : \forall\bar{\mathsf{X}}.\mathsf{T}}{\Gamma \vdash_{H/M} e : \mathsf{T}}$$

We add the base rule (H/M-BASE) to introduce constants and their associated types in Hindley/Milner thus comparing coherently the BQ family to Hindley/Milner.

4.2 Comparison

We now compare our type system with Hindley/Milner. We show that every type derivation in Hindley/Milner is also derivable in BQ^{sub,abs} (theorem 4.2.1) and later give a direct comparison between our algorithmic presentation BQ and Hindley/Milner (theorem 4.2.2).

Theorem 4.2.1 *If* $\Gamma \vdash_{H/M} e : S$ *then* $[\![\Gamma]\!]_{H/M \triangleright BQ} \vdash_p e : [\![S]\!]_{H/M \triangleright BQ}$ *in* BQ^{sub,abs}.

Proof (theorem 4.2.2) By induction on the Hindley/Milner typing rules.
$\boxed{\text{(H/M-BASE)}}$ Immediate since $[\![\mathcal{B}(c)]\!]_{\mathsf{H/M \triangleright BQ}} = \forall\emptyset|\mathsf{true}.\mathcal{B}(c)$.

$\boxed{\text{(H/M-VAR)}}$ Suppose $\Gamma \vdash_{\mathsf{H/M}} x : \mathsf{S}$. By rule hypothesis $\Gamma(x) = \mathsf{S}$ and by definition $[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}}(x) = [\![\mathsf{S}]\!]_{\mathsf{H/M \triangleright BQ}}$. By applying the (BQ-SD-VAR) rule we have $[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} x : [\![\mathsf{S}]\!]_{\mathsf{H/M \triangleright BQ}}$.

$\boxed{\text{(H/M-ABS)}}$ Suppose $\Gamma \vdash_{\mathsf{H/M}} \lambda x.e : \mathsf{T}_1 \to \mathsf{T}_2$. By rule hypothesis $\Gamma \cdot x : \mathsf{T}_1 \vdash_{\mathsf{H/M}} e : \mathsf{T}_2$ and by induction

$$\underbrace{[\![\Gamma \cdot x : \mathsf{T}_1]\!]_{\mathsf{H/M \triangleright BQ}}}_{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \cdot x : \forall\emptyset|\mathsf{true}.\mathsf{T}_1} \vdash_{\mathsf{p}} e : \underbrace{[\![\mathsf{T}_2]\!]_{\mathsf{H/M \triangleright BQ}}}_{\forall\emptyset|\mathsf{true}.\mathsf{T}_2}$$

we can derive

$$\dfrac{\dfrac{\dfrac{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \cdot x : \forall\emptyset|\mathsf{true}.\mathsf{T}_1 \vdash_{\mathsf{p}} e : \forall\emptyset|\mathsf{true}.\mathsf{T}_2}{\mathsf{true}; [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \cdot x : \forall\emptyset|\mathsf{true}.\mathsf{T}_1 \vdash_{\mathsf{m}} e : \mathsf{T}_2}\,\text{(BQ-SD-START)}}{\mathsf{true}; [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{m}} \lambda x.e : \mathsf{T}_1 \to \mathsf{T}_2}\,\text{(BQ-DEC-ABS)} \qquad \models \mathsf{true}}{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} \lambda x.e : \forall\bar{\mathsf{X}}|\mathsf{true}.\mathsf{T}_1 \to \mathsf{T}_2}\,\text{(BQ-SD-FIN)}$$

where $\bar{\mathsf{X}} = \mathrm{ftv}(\mathsf{T}_1 \to \mathsf{T}_2) \cup \mathrm{ftv}(\mathsf{true}) \setminus \mathrm{ftv}([\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}})$. By lemma (lemma 2.8.5)

$$\forall\bar{\mathsf{X}}|\mathsf{true}.\mathsf{T}_1 \to \mathsf{T}_2 <: \forall\emptyset|\mathsf{true}.\mathsf{T}_1 \to \mathsf{T}_2 = [\![\mathsf{T}_1 \to \mathsf{T}_2]\!]_{\mathsf{H/M \triangleright BQ}}$$

and we can conclude by applying the (BQ-DEC-SUB) rule.

$\boxed{\text{(H/M-APP)}}$ Suppose $\Gamma \vdash_{\mathsf{H/M}} e_1\ e_2 : \mathsf{T}_2$. By rule hypothesis $\Gamma \vdash_{\mathsf{H/M}} e_1 : \mathsf{T}_1 \to \mathsf{T}_2$ and $\Gamma \vdash_{\mathsf{H/M}} e_2 : \mathsf{T}_1$. By induction

$$[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e_1 : \underbrace{[\![\mathsf{T}_1 \to \mathsf{T}_2]\!]_{\mathsf{H/M \triangleright BQ}}}_{\forall\emptyset|\mathsf{true}.\mathsf{T}_1 \to \mathsf{T}_2} \qquad [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e_2 : \underbrace{[\![\mathsf{T}_1]\!]_{\mathsf{H/M \triangleright BQ}}}_{\forall\emptyset|\mathsf{true}.\mathsf{T}_1}$$

and we can derive

$$\dfrac{\dfrac{\dfrac{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e_1 : \forall\emptyset|\mathsf{true}.\mathsf{T}_1 \to \mathsf{T}_2}{\mathsf{true}; [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{m}} e_1 : \mathsf{T}_1 \to \mathsf{T}_2}\,\text{(BQ-SD-START)} \quad \dfrac{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e_2 : \forall\emptyset|\mathsf{true}.\mathsf{T}_1}{\mathsf{true}; [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{m}} e_2 : \mathsf{T}_1}\,\text{(BQ-SD-START)}}{\overbrace{\mathsf{true} \wedge \mathsf{true} \wedge \mathsf{T}_1 \to \mathsf{T}_2 <: \mathsf{X} \to \mathsf{Y} \wedge \mathsf{T}_1 <: \mathsf{X}}^{C}; [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{m}} e_1\ e_2 : \mathsf{Y}}\,\text{(BQ-SD-APP)} \qquad \models C}{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e_1\ e_2 : \forall\bar{\mathsf{X}}|C.\mathsf{Y}}\,\text{(BQ-SD-FIN)}$$

where $\bar{\mathsf{X}} = \mathrm{ftv}(\mathsf{Y}) \cup \mathrm{ftv}(C) \setminus \mathrm{ftv}([\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}})$. As $\forall\bar{\mathsf{X}}|C.\mathsf{Y} <: \forall\emptyset|\mathsf{true}.\mathsf{T}_2 = [\![\mathsf{T}_2]\!]_{\mathsf{H/M \triangleright BQ}}$, we can conclude by applying the (BQ-DEC-SUB) rule.

$\boxed{\text{(H/M-LET)}}$ Suppose $\Gamma \vdash_{\mathsf{H/M}} \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \mathsf{T}$. By rule hypothesis $\Gamma \vdash_{\mathsf{H/M}} e_1 : \mathsf{S}$ and $\Gamma \cdot x : \mathsf{S} \vdash_{\mathsf{H/M}} e_2 : \mathsf{T}$. By induction

$$[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e_1 : [\![\mathsf{S}]\!]_{\mathsf{H/M \triangleright BQ}} \qquad \underbrace{[\![\Gamma \cdot x : \mathsf{S}]\!]_{\mathsf{H/M \triangleright BQ}}}_{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \cdot x : [\![\mathsf{S}]\!]_{\mathsf{H/M \triangleright BQ}}} \vdash_{\mathsf{p}} e_2 : \underbrace{[\![\mathsf{T}]\!]_{\mathsf{H/M \triangleright BQ}}}_{\forall \emptyset | \mathsf{true}.\mathsf{T}}$$

we can derive

$$\textsc{(BQ-sd-fin)}\ \dfrac{\textsc{(BQ-sd-let)}\ \dfrac{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e_1 : [\![\mathsf{S}]\!]_{\mathsf{H/M \triangleright BQ}} \qquad \dfrac{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \cdot x : [\![\mathsf{S}]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e_2 : \forall \emptyset | \mathsf{true}.\mathsf{T}}{\mathsf{true}; [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \cdot x : [\![\mathsf{S}]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{m}} e_2 : \mathsf{T}}\ \textsc{(BQ-sd-start)}}{\mathsf{true}; [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{m}} \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \mathsf{T}} \qquad \models \mathsf{true}}{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \forall \bar{\mathsf{X}} | \mathsf{true}.\mathsf{T}}$$

where $\bar{\mathsf{X}} = \mathrm{ftv}\,(\mathsf{T}) \cup \mathrm{ftv}(\mathsf{true}) \setminus \mathrm{ftv}\,([\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}})$. By lemma (lemma 2.8.5)

$$\forall \bar{\mathsf{X}} | \mathsf{true}.\mathsf{T} <: \forall \emptyset | \mathsf{true}.\mathsf{T} = [\![\mathsf{T}]\!]_{\mathsf{H/M \triangleright BQ}}$$

and we can conclude by applying the (BQ-DEC-SUB) rule.

$\boxed{\textsc{(H/M-gen)}}$ Suppose $\Gamma \vdash_{\mathsf{H/M}} e : \forall \bar{\mathsf{X}}.\mathsf{T}$. Without loss of generality, we suppose that the conversion of $\forall \bar{\mathsf{X}}.\mathsf{T}$ does not mention type variables vacuously (lemma 2.8.6). By rule hypothesis $\Gamma \vdash_{\mathsf{H/M}} e : \mathsf{T}$ and $\bar{\mathsf{X}}\ \#\ \mathrm{ftv}(\Gamma)$ and by induction $[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e : [\![\mathsf{T}]\!]_{\mathsf{H/M \triangleright BQ}}$. We can derive

$$\textsc{(BQ-sd-fin)}\ \dfrac{\textsc{(BQ-sd-start)}\ \dfrac{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{p}} e : [\![\mathsf{T}]\!]_{\mathsf{H/M \triangleright BQ}}}{\mathsf{true}; [\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{m}} e : \mathsf{T}} \qquad \models \mathsf{true}}{[\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}} \vdash_{\mathsf{m}} e : \forall \bar{\mathsf{Y}} | \mathsf{true}.\mathsf{T}}$$

where $\bar{\mathsf{Y}} = \mathrm{ftv}\,(\mathsf{T}) \cup \mathrm{ftv}(\mathsf{true}) \setminus \mathrm{ftv}\,([\![\Gamma]\!]_{\mathsf{H/M \triangleright BQ}})$. By (lemma 4.1.1) we know that $\bar{\mathsf{X}} \subseteq \bar{\mathsf{Y}}$ and by (lemma 2.8.5)

$$\forall \bar{\mathsf{Y}} | \mathsf{true}.\mathsf{T} <: \forall \bar{\mathsf{X}} | \mathsf{true}.\mathsf{T} = [\![\forall \bar{\mathsf{X}}.\mathsf{T}]\!]_{\mathsf{H/M \triangleright BQ}}$$

which allows us to conclude by applying the (BQ-DEC-SUB) rule.

$\boxed{\textsc{(H/M-inst)}}$ Immediate since $[\![\forall \bar{\mathsf{X}}.\mathsf{T}]\!]_{\mathsf{H/M \triangleright BQ}} <: [\![\mathsf{T}]\!]_{\mathsf{H/M \triangleright BQ}}$.

Theorem 4.2.2 *If $\Gamma \vdash_{H/M} e : S$ then $[\![\Gamma]\!]_{H/M \triangleright BQ} \vdash_p e : S'$ in BQ with $S' <: [\![S]\!]_{H/M \triangleright BQ}$.*

Proof (theorem 4.2.2) Follows directly from (corollary 3.2.4) and (theorem 4.2.1).

Chapter 5

More on Constraints

5.1 Efficiency of Constraint Solving

We now concentrate on the efficiency of constraint solving. Building upon previous algebraic results [5, 30, 37], we show that constraint satisfiability in presence of positive subtyping is PTIME-complete. Benke [5] showed that satisfiability of inequalities in Helly posets (definition 5.1.4) is decidable in PTIME. To reuse this result, our analysis defines the type poset $\mathcal{T}$ as a term algebra of functions and records constructed from atomic types (definition 5.1.10) and show that the term algebra construction preserves the Helly property.

We begin our analysis by defining fences, distances, disks and the Helly property. Quilliot [31] introduced these definitions; Nevermann/Rival [19] and Benke [5] later reformulated them. Intuitively, a fence is an oscillating path within a poset, each oscillation corresponding to a shift in the direction the path takes along the links in the poset. The distance between two elements in a poset is the number of oscillations in the smallest fence that links the two elements. A disk of a given size **n** around an element T is the set of elements at distance less than **n** from T.

Introducing positive functional subtyping over an arbitrary poset preserves all structural properties. In effect, positive functional subtyping lifts the subtyping relation of the poset but does not alter its shape. In particular, introducing positive functional subtyping preserves the Helly property. Contra-variant functional subtyping, in comparison, greatly modifies the shape of posets (Figure 5.1). Hoang and Mitchell [10] studied the latter case and showed that satisfiability of inequalities in that context can be PSPACE-hard.

Since our base type poset is trivial and positive functional subtyping simply lifts the underlying subtyping relation, we need to introduce non-trivial subtyping. We have chosen to do so using records and show that record subtyping preserves the Helly property. This approach differs from Hoang and Mitchell [10], who consider a complex base type poset and introduce contra-variant functional subtyping. Finally, note that our results would hold over any base type poset that has the Helly property, which includes lattices and trees.

Definition 5.1.1 *An up-fence of length n from T_1 to T_2 in P, written $F^+_P(\mathsf{T}_1, \mathsf{T}_2)$, is a set $\{\mathsf{U}_0, \ldots, \mathsf{U}_n\}$ such that $\mathsf{T}_1 = \mathsf{U}_0$ and $\mathsf{T}_2 = \mathsf{U}_n$ such that $\mathsf{U}_{2i} <: \mathsf{U}_{2i+1} :> \mathsf{U}_{2i+2}$. A down fence $F^-_P(\mathsf{T}_1, \mathsf{T}_2)$ is defined equivalently with $\mathsf{U}_{2i} :> \mathsf{U}_{2i+1} <: \mathsf{U}_{2i+2}$.*

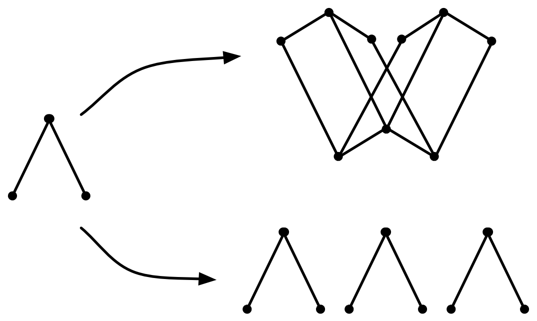

Figure 5.1: Positive Subtyping vs. Contra-variant Subtyping – one arrow posets are shown.

An up-fence between T_1 and T_2 could be a down-fence from T_2 to T_1. The direction of a fence is not inherent to its structure, but rather a point of view. In the next section, we explore precisely this duality.

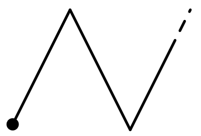

Figure 5.2: An up-fence.

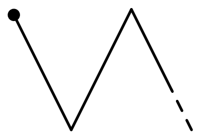

Figure 5.3: A down-fence.

Definition 5.1.2 *The up-distance $d_P^+(T_1, T_2)$ of two elements in a poset P is the smallest integer n such that there exists an up-fence of length n from T_1 to T_2. The down-distance is defined dually. Finally, the distance T_1 to T_2 is defined as*

$$d_P(T_1, T_2) = (d_P^+(T_1, T_2), d_P^-(T_1, T_2)).$$

Definition 5.1.3 *A disk of center T and of radius $\mathbf{n} = (n^+, n^-)$ in P is the set*

$$D_P(T, \mathbf{n}) = \{U \in P \mid d_P(T, U) \leq \mathbf{n}\}$$

that is, the set of all U satisfying the two inequatlities $d_P^+(T, U) \leq n^+$ and $d_P^-(T, U) \leq n^-$.

Definition 5.1.4 *An ordered set satisfies the two-disk property (also known as the Helly property) if, for every family $\mathcal{D}$ of disks,*

$$\bigcap \mathcal{D} \neq \emptyset$$

whenever $D_1 \cap D_2 \neq \emptyset$ for each D_1, D_2 in $\mathcal{D}$.

Poset with Arrow

We consider a partially ordered set (or poset) which we extend by adding positive subtyping. As mentioned earlier, Hoang and Mitchell [10] consider this case in depth for contra-variant subtyping.

Definition 5.1.5 *Let $\langle P, <:_P\rangle$ be a partially ordered set. We denote by $P^{\rightarrow}$ the term algebra over $(P, \rightarrow)$. It is partially ordered by extending the $<:_P$ using the* (BQ-S-FUN) *rule.*

We define an equivalence relation that partitions types of $P^{\rightarrow}$ based on their arrow structure. This relation requires invariant types to be equal and considers co-variant types as equivalent.

Definition 5.1.6 *We define the relation $\sim$ by*

$$\text{(EQ-BASE)}\ \frac{T_1 \in P \qquad T_2 \in P}{T_1 \sim T_2} \qquad\qquad \text{(EQ-ARROW)}\ \frac{T_2 \sim T_2'}{T_1 \rightarrow T_2 \sim T_1 \rightarrow T_2'}$$

and we say that T_1 has the same arrow structure as T_2 whenever $T_1 \sim T_2$.

The following theorem captures the properties of the $\sim$ relation.

Theorem 5.1.1 *$\sim$ is an equivalence relation. We denote by $[T]_\sim$ the equivalence class of T under $\sim$.*

We now capture the equality of invariant types under the equivalence relation.

Theorem 5.1.2 *If $T_1 \rightarrow T_1'$ and $T_2 \rightarrow T_2'$ are elements of $[T]_\sim$, then $T_1 = T_2$.*

To exploit the equivalence relation and link it with the subtyping relation, we define an extraction function which selects the co-variant portion of a type.

Definition 5.1.7 *We define the function* extract *by*

$$\text{(EX-BASE)}\ \frac{T \in P}{\text{extract}(T) = T} \qquad\qquad \text{(EX-ARROW)}\ \frac{\text{extract}(T_2) = T_2'}{\text{extract}(T_1 \rightarrow T_2) = T_2'}$$

which extracts the lower right leaf of a type.

The extraction function maps types in $P^{\rightarrow}$ to types in P.

Theorem 5.1.3 *The range of* extract *is P.*

The invariance of the argument type of a functional type makes subtyping in $P^{\rightarrow}$ reducible to subtyping in P, and by extension each equivalence class $[T]_\sim$ is isomorphic to P (theorem 5.1.5).

Theorem 5.1.4 *If T_1 and T_2 are elements of $[T]_\sim$ then $T_1 <: T_2$ if and only if* $\text{extract}(T_1) <: \text{extract}(T_2)$.

Proof (theorem 5.1.4) By induction on T_2.
$\boxed{T_2 \in P}$ By normal form (theorem 2.6.1), $T_1 \in P$ and we can conclude since $\text{extract}(T_1) = T_1$ and $\text{extract}(T_2) = T_2$.

$\boxed{T_2 = T_{12} \rightarrow T_2'}$

[⇒] We suppose that $\mathsf{T}_1 <: \mathsf{T}_2$. By normal form (theorem 2.6.1), $\mathsf{T}_1 = \mathsf{T}_{12} \to \mathsf{T}'_1$ and $\mathsf{T}'_1 <: \mathsf{T}'_2$. By induction $\mathrm{extract}(\mathsf{T}'_1) <: \mathrm{extract}(\mathsf{T}'_2)$ and thus $\mathrm{extract}(\mathsf{T}_1) <: \mathrm{extract}(\mathsf{T}_2)$.

[⇐] We suppose that $\mathrm{extract}(\mathsf{T}_1) <: \mathrm{extract}(\mathsf{T}_2)$ and we conclude using (theorem 5.1.2).

Theorem 5.1.5 $[T]_\sim$ *is isomorphic to P.*

Proof (theorem 5.1.5) Let T be in $P^{\to}$. We suppose that it is not in P since this case is trivial. Define the map $\varphi(\mathsf{U}) = \mathrm{extract}(\mathsf{U})$ between $[\mathsf{T}]_\sim$ and P. We have shown that

$$\varphi(\mathsf{U}_1) <: \varphi(\mathsf{U}_2) \iff \mathsf{U}_1 <: \mathsf{U}_2$$

and since φ is one-to-one and onto, φ is in fact an isomorphism.

Corollary 5.1.1 *$P^{\to}$ is partitioned by $\sim$ into isomorphic copies of P.*

Theorem 5.1.6 *If P satisfies the Helly property so does $P^{\to}$.*

Proof (theorem 5.1.6) Immediate since $P^{\to}$ is partitioned into isomorphic copies of P.

5.1.1 Poset with Records

In this section, we prove that the structural subtyping introduced by records preserves the Helly property. We build a term algebra for records (definition 5.1.8) and give an inductive proof.

Definition 5.1.8 *Let $\langle P, <:_P\rangle$ be a partially ordered set. We define*

$$\begin{array}{rcl} P^{\{\}(0)} & = & P \\ P^{\{\}(k+1)} & = & P^{\{\}(k)} \cup \displaystyle\bigcup_{\{l_1,\ldots,l_n\}\subset\mathcal{L}} \left\{l_1 : P^{\{\}(k)}, \ldots, l_n : P^{\{\}(k)}\right\} \\ P^{\{\}} & = & \displaystyle\bigcup_{k\geq 0} P^{\{\}(k)} \end{array}$$

and we extend the partial ordering $<:_P$ using the (BQ-S-REC) *rule.*

Given two types, the up-distance between them is at most 2 since every type is a subtype of top, and by symmetry their down-distance is at most 3.

Theorem 5.1.7 *The maximal up-distance in $P^{\{\}}$ is* 2*, and the maximal down-distance is* 3.

Proof (theorem 5.1.7) Follows from the presence of $\top$.

Based on the previous observation and on the definition of distance, we can give a simpler presentation of disks in $P^{\{\}}$.

Theorem 5.1.8 *Disks $D_{P^{\{\}}}(T, \mathbf{n})$ have the shapes outlined in Figure 5.4.*

$\mathbf{n}$	0	1	2
0	$\{\mathsf{T}\}$	$\{\mathsf{T}\}$	$\{\mathsf{T}\}$
1	$\{\mathsf{T}\}$	$\{\mathsf{T}\}$	$\downarrow\mathsf{T}$
2	$\{\mathsf{T}\}$	$\uparrow\mathsf{T}$	$\uparrow\mathsf{T}\cup\downarrow\mathsf{T}$
3	$\{\mathsf{T}\}$	$\uparrow\mathsf{T}$	$P^{\{\}}$

Figure 5.4: Disks in $P^{\{\}}$.

Proof (theorem 5.1.8) We show three cases, the others are similar.

$$\begin{aligned} D_{P^{\{\}}}(\mathsf{T},(1,1)) &= \left\{\mathsf{U}\,\middle|\, d^{+}(\mathsf{T},\mathsf{U}) \le 1 \wedge d^{-}(\mathsf{T},\mathsf{U}) \le 1\right\} \\ &= \{\mathsf{U} \mid \mathsf{T} <: \mathsf{U} \wedge \mathsf{U} <: \mathsf{T}\} \\ &= \{\mathsf{T}\} \end{aligned}$$

$$\begin{aligned} D_{P^{\{\}}}(\mathsf{T},(2,1)) &= \left\{\mathsf{U}\,\middle|\, d^{+}(\mathsf{T},\mathsf{U}) \le 2 \wedge d^{-}(\mathsf{T},\mathsf{U}) \le 1\right\} \\ &= \left\{\mathsf{U}\,\middle|\, \exists \mathsf{U}' : \mathsf{T} <: \mathsf{U}' \wedge \mathsf{U} <: \mathsf{U}' \wedge \mathsf{U} <: \mathsf{T}\right\} \\ &= \downarrow\mathsf{T} \end{aligned}$$

$$\begin{aligned} D_{P^{\{\}}}(\mathsf{T},(2,2)) &= \left\{\mathsf{U}\,\middle|\, d^{+}(\mathsf{T},\mathsf{U}) \le 2 \wedge d^{-}(\mathsf{T},\mathsf{U}) \le 2\right\} \\ &= \left\{\mathsf{U}\,\middle|\, \exists \mathsf{U}',\mathsf{U}'' : \mathsf{T} <: \mathsf{U}' \wedge \mathsf{U} <: \mathsf{U}' \wedge \mathsf{U}'' <: \mathsf{T} \wedge \mathsf{U}'' <: \mathsf{U}\right\} \\ &= \uparrow\mathsf{T}\cup\downarrow\mathsf{T} \end{aligned}$$

Theorem 5.1.9 *If P satisfies the Helly property so does $P^{\{\}}$.*

Proof (theorem 5.1.9) We suppose that P satisfies the Helly property, and we prove that $P^{\{\}}$ satisfies it by induction on $P^{\{\}}$.
$\boxed{P^{\{\}(0)}}$ Immediate.

$\boxed{P^{\{\}(k+1)}}$ By induction, $P^{\{\}}(k)$ satisfies the Helly property. We only consider disks of radius $(2, 1)$ since the others contain $\top$ or intersect trivially. Take a family of disks $\mathcal{D}$ such that for each $D_1, D_2 \in \mathcal{D}$ we have $D_1 \cap D_2 \neq \emptyset$. For each pair of disk $D_1 = \downarrow\mathsf{T}_1$ and $D_2 = \downarrow\mathsf{T}_2$ we have $\mathsf{T}_{12} <: \mathsf{T}_1$ and $\mathsf{T}_{12} <: \mathsf{T}_2$. The two types T_1, T_2 must either be both in $P^{\{\}(k)}$ or both in $P^{\{\}(k+1)}$, because of the subtyping relation. The former case being trivial, we concentrate on the latter. By normal form (theorem 2.6.1) T_{12} contains at least the union of T_1 and T_2's labels. Since these observations have been made on arbitrary disks, we can construct a type $\mathsf{T}_\perp$ of the union of all the

T_i's labels. Depth subtyping can be approached with a similar reasoning, if T_i and T_j have intersecting labels then we have a common subtype.

Note that when considering a whole expression, the number of labels is finite and we thus need not worry about infinite family of disks.

5.1.2 Poset with Arrow and Record

We now combine the previous results and use the $\rightarrow$ and $\{\}$ constructions to iteratively construct a type poset (definition 5.1.9). We then show that this iterative construction produces the type poset $\mathcal{T}$ (theorem 5.1.10) and can therefore conclude that our types satisfy the Helly property.

Definition 5.1.9 *Let $\langle P, <:_P\rangle$ be a partially ordered set. We define*

$$\begin{array}{rcl} P^{\rightarrow,\{\}(0)} & = & P \\ P^{\rightarrow,\{\}(k+1)} & = & \left(\left(P^{\rightarrow,\{\}(k)}\right)^{\rightarrow}\right)^{\{\}} \\ P^{\rightarrow,\{\}} & = & \bigcup\limits_{k\geq 0} P^{\rightarrow,\{\}(k)}. \end{array}$$

Definition 5.1.10 *Let $\mathcal{A}$ denote the set of atomic types defined as the union of base types, type variables $\mathcal{V}$ and the top type.*

Theorem 5.1.10 $\mathcal{A}^{\rightarrow,\{\}} = \mathcal{T}$.

Proof (theorem 5.1.10) By double inclusion.
$\boxed{\subseteq}$ Direct since $\mathcal{T}$ is the term algebra from which $\mathcal{A}$ is constructed.

$\boxed{\supseteq}$ By induction on $\mathsf{T} \in \mathcal{T}$.
[$\top$, B, X] Immediate since these types are in $\mathcal{A}$.

[$\mathsf{T}_1 \rightarrow \mathsf{T}_2$] By induction $\mathsf{T}_1 \in \mathcal{A}^{\rightarrow,\{\}}$ and $\mathsf{T}_2 \in \mathcal{A}^{\rightarrow,\{\}}$. Since T_1 and T_2 are finite terms, there exists n_1 and n_2 such that $\mathsf{T}_1 \in \mathcal{A}^{\rightarrow,\{\}(n_1)}$ and $\mathsf{T}_2 \in \mathcal{A}^{\rightarrow,\{\}(n_2)}$. Let $n = \max(n_1, n_2)$, we have $\mathsf{T}_1 \in \mathcal{A}^{\rightarrow,\{\}(n)}$ and $\mathsf{T}_2 \in \mathcal{A}^{\rightarrow,\{\}(n)}$, and we show that $\mathsf{T} = \mathsf{T}_1 \rightarrow \mathsf{T}_2 \in \mathcal{A}^{\rightarrow,\{\}(n+1)}$

$$\begin{array}{rcl} \mathcal{A}^{\rightarrow,\{\}(n+1)} & = & \left(\left(P^{\rightarrow,\{\}(n)}\right)^{\rightarrow}\right)^{\{\}} \\ & \supseteq & \left(P^{\rightarrow,\{\}(n)}\right)^{\rightarrow} \\ & \supseteq & P^{\rightarrow,\{\}(n)} \rightarrow P^{\rightarrow,\{\}(n)} \end{array}$$

since $\mathsf{T} \in P^{\rightarrow,\{\}(n)} \rightarrow P^{\rightarrow,\{\}(n)}$ we have by inclusion that $\mathsf{T} \in \mathcal{A}^{\rightarrow,\{\}(n+1)}$ and a fortiori $\mathsf{T} \in \mathcal{A}^{\rightarrow,\{\}}$ concluding this case.

[$\left\{\vec{l} : \vec{\mathsf{T}_l}\right\}$] Exact same reasoning as for the previous case.

Theorem 5.1.11 *$\mathcal{A}$ satisfies the Helly property.*

Proof (theorem 5.1.11) Immediate since $\mathcal{A}$ is a tree.

Corollary 5.1.2 *$\mathcal{T}$ satisfies the Helly property.*

Proof (corollary 5.1.2) $\mathcal{A}^{\rightarrow,\{\}}$ is a Helly poset since $\mathcal{A}$ satisfies the Helly property (theorem 5.1.11) and $\rightarrow$, $\{\}$ are Helly property preserving (theorem 5.1.6), (theorem 5.1.9). In addition, we have shown that $\mathcal{A}^{\rightarrow,\{\}} = \mathcal{T}$ (theorem 5.1.10).

Corollary 5.1.3 *Satisfiability of a system of inequalities in $\mathcal{T}$ is decidable in PTIME.*

Proof (corollary 5.1.2) Follows from Benke [5].

5.2 Satisfiability of Constraints

Benke [5] showed that a Helly poset is satisfiable if and only if it is weakly satisfiable (definition 5.2.1) and distance consistent (definition 5.2.2). Since our types form a Helly poset, we can apply this result with minor changes to our work. In this section, we describe weak satisfiability and the distance consistency inference system Benke [5] defined and discuss its use in our type inference system.

We consider a subset of the constraint language presented earlier, namely subtyping constraints ($\mathsf{T}_1 <: \mathsf{T}_2$) and logical conjunction ($C_1 \wedge C_2$). Constraints can therefore be seen as a set of subtyping constraints. Removing the true predicate does not modify the satisfiability of constraints, and the false predicate cannot appear in a satisfiable set of constraints. In addition, the existential quantifier can be floated out. Finally, we solve constraints in $\mathcal{M}$, eliminating universally quantified type variables. Since we are concerned with satisfiability (definition 2.7.5), this restriction does not impact the expressiveness of our type system.

Definition 5.2.1 *A constraint C is weakly satisfiable if the constraint C' obtained by replacing subtyping constraints by equality constraints and mapping base types to a constant $\star$ is satisfiable.*

For the sake of weak satisfiability, record equality compares common labels only and requires one record type's labels to be a subset of the other's record type. Weak satisfiability can be seen as a shape consistency and is a necessary condition to satisfiability. Weak satisfiability can be verified using unification and is therefore PTIME-complete [24, 18].

5.2.1 Distance Consistency

We define a two-place predicate $C \vdash_{\mathsf{d}} d^{\epsilon}(\mathsf{T}_1 <: \mathsf{T}_2) \leq n$ which can be read as "if C is satisfiable, the ϵ-distance between T_1 and T_2 must be less than or equal to n" where ϵ is either $+$ for up-distance or $-$ for down-distance.

$$\text{(DC-AX)}\ \frac{\mathsf{T}_1 <: \mathsf{T}_2 \in C}{C \vdash_{\mathsf{d}} d^{+}(\mathsf{T}_1, \mathsf{T}_2) \leq 1} \qquad \text{(DC-SYMM)}\ \frac{C \vdash_{\mathsf{d}} d^{\epsilon}(\mathsf{T}_1, \mathsf{T}_2) \leq n}{C \vdash_{\mathsf{d}} d^{\epsilon(-1)^n}(\mathsf{T}_2, \mathsf{T}_1) \leq n} \qquad \text{(DC-DUAL)}\ \frac{C \vdash_{\mathsf{d}} d^{\epsilon}(\mathsf{T}_1, \mathsf{T}_2) \leq n}{C \vdash_{\mathsf{d}} d^{-\epsilon}(\mathsf{T}_1, \mathsf{T}_2) \leq n+1}$$

$$\text{(DC-JOIN)}\ \frac{C \vdash_{\mathsf{d}} d^{\epsilon}(\mathsf{T}_1, \mathsf{T}_2) \leq n \qquad C \vdash_{\mathsf{d}} d^{\epsilon(-1)^{n-1}}(\mathsf{T}_2, \mathsf{T}_3) \leq m \qquad k = \begin{cases} m & n = 0 \\ n & m = 0 \\ n+m-1 & \text{otherwise} \end{cases}}{C \vdash_{\mathsf{d}} d^{\epsilon}(\mathsf{T}_1, \mathsf{T}_3) \leq k}$$

$$\text{(DC-FUN)}\ \frac{C \vdash_{\mathsf{d}} d^{\epsilon}(\mathsf{T}_1 \to \mathsf{T}_2, \mathsf{T}_1' \to \mathsf{T}_2') \leq n \qquad n \in D_{\epsilon} \text{ where } D_{+} = \{1\} \text{ and } D_{-} = \{1, 2\}}{C \vdash_{\mathsf{d}} d^{+/-}(\mathsf{T}_1, \mathsf{T}_1') = 0 \qquad C \vdash_{\mathsf{d}} d^{\epsilon}(\mathsf{T}_2 \mathsf{T}_2') \leq n}$$

$$\text{(DC-REC)}\ \frac{C \vdash_{\mathsf{d}} d^{\epsilon}(\{\vec{l_-} : \vec{\mathsf{T}}_-\}, \{\vec{l_+} : \vec{\mathsf{T}}_+\}) \leq n}{C \vdash_{\mathsf{d}} d^{\epsilon}(\mathsf{T}_-^l, \mathsf{T}_+^l) \leq n \qquad l \in \bar{l}_{\epsilon}}$$

The axiomatic rule (DC-AX) transforms a subtyping constraint into an up-distance constraint. If T_1 is a subtype of T_2, then we can draw a unit up-fence between them. The symmetry rule (DC-SYMM) is a combination of two cases. In the first case, if the up-distance (respectively down-distance) between T_1 and T_2 is even, the fence joining T_1 and T_2 is symmetrical; therefore we can constrain the up-distance (respectively down-distance) between T_2 and T_1. Figure 5.5 illustrates this graphical symmetry. In the second case, if the up-distance (respectively down-distance) between T_1 and T_2 is odd, the fence's edges are in two different directions. Consequently, we constrain the down-distance (respectively up-distance). We illustrate this case in Figure 5.6. The dual rule

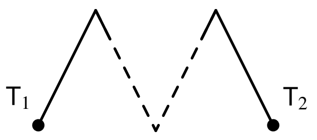

Figure 5.5: (DC-SYMM) with an even distance. $d^{+}(\mathsf{T}_1, \mathsf{T}_2)$ can be reversed to derive $d^{+}(\mathsf{T}_2, \mathsf{T}_1)$.

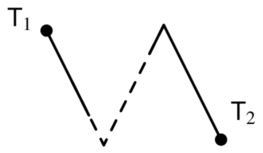

Figure 5.6: (DC-SYMM) with an odd distance. $d^{-}(\mathsf{T}_1, \mathsf{T}_2)$ can be reversed to derive $d^{+}(\mathsf{T}_2, \mathsf{T}_1)$.

(DC-DUAL) constrains the inverse distance between two types. Knowing that T_2 is reachable from T_1, using an n distance fence, we can derive that an $n+1$ distance fence joins T_1 and T_2 in the inverse direction. Indeed, the worst scenario will lead us to draw a unit fence from T_1 to itself and reuse the fence previously known.

The join rule (DC-JOIN) is a special case of subtyping transitivity. If a fence of distance n exists between T_1 and T_2 and a fence of distance m exists between T_2 and T_3, then we can combine these fences to form an $n+m$

distance fence. The join rule captures the specific situation when the direction of the two fences coincides at the join point. In this case, the distance of the combined fence is $n + m - 1$. For instance, if we have a down-fence between T_1 and T_2 with an odd distance, the direction of the last link will be down. Therefore, if we combine this fence with a down-fence from T_2 to T_3 the two links at the join point can be combined into a single one. We illustrate this reasoning in Figure 5.7. In the special case where one or both of the distances is 0, this rule falls back on subtyping's transitivity. We have adapted the functional sub-term rule for positive subtyping and

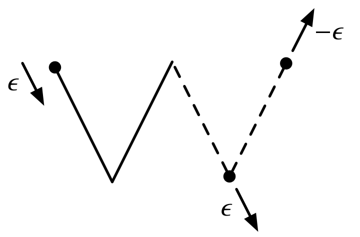

Figure 5.7: (DC-JOIN)

added the record sub-term rule. The functional sub-term rule (DC-FUN) captures the invariance of argument types by equating the two argument types and propagates the co-variance of the return type. In the record sub-term rule (DC-REC), the co-variance of the labels' types is propagated. We use a notational shortcut to differentiate between up and down distance. The up-distance regards the first type as a subtype of the second, whereas the down-distance regards the first type as a supertype of the second; thus if the up-distance between two records is observed ($\epsilon = +$), the record $\left\{\vec{l}_- : \vec{\mathsf{T}}_-\right\}$ ought to have more fields than $\left\{\vec{l}_+ : \vec{\mathsf{T}}_+\right\}$. We can therefore conclude by constraining all labels in $l_+ = l_\epsilon$. The same reasoning applies for the other case.

Definition 5.2.2 *A constraint C is said to be distance consistent whenever for each T_1, T_2 in $\mathcal{M}$ and for a natural number n if $C \vdash_d d^\epsilon(T_1, T_2) \leq n$ then $d^\epsilon_{\mathcal{M}}(T_1, T_2) \leq n$.*

Distance consistency of a constraint C can be verified by constructing two $O(|C|) \times O(|C|)$ arrays containing the up and down-distances between ground types. The inference system described before can then be used to find inconsistencies, if any, in C. The up and down-distances between ground types can be straightforwardly calculated, and we summarize the results here.

$$d^+(\mathsf{T}, \top) = \begin{cases} 0 & \mathsf{T} = \top \\ 1 & \text{otherwise} \end{cases} \qquad d^+(\mathsf{B}_1, \mathsf{B}_2) = \begin{cases} 0 & \mathsf{B}_1 = \mathsf{B}_2 \\ 2 & \mathsf{B}_1 \neq \mathsf{B}_2 \end{cases}$$

$$d^+(\mathsf{T}_1 \to \mathsf{T}_2, \mathsf{T}'_1 \to \mathsf{T}'_2) = \begin{cases} d^+(\mathsf{T}_2, \mathsf{T}'_2) & \mathsf{T}_1 = \mathsf{T}'_1 \\ 2 & \mathsf{T}_1 \neq \mathsf{T}'_1 \end{cases}$$

$$d^+(\left\{\vec{l}_1 : \vec{\mathsf{T}}_1\right\}, \left\{\vec{l}_2 : \vec{\mathsf{T}}_2\right\}) = \begin{cases} 0 & \left\{\vec{l}_1 : \vec{\mathsf{T}}_1\right\} = \left\{\vec{l}_2 : \vec{\mathsf{T}}_2\right\} \\ 1 & \bar{l}_2 \subseteq \bar{l}_1 \text{ and } \max_{l \in \bar{l}_2} d^+(\mathsf{T}^l_1, \mathsf{T}^l_2) \leq 1 \\ 2 & \text{otherwise} \end{cases}$$

All other cases have an up-distance of 2. The down-distance can be similarly calculated.

$$d^{-}(\top,\mathsf{T}) = \begin{cases} 0 & \mathsf{T} = \top \\ 1 & \text{otherwise} \end{cases} \qquad d^{-}(\mathsf{T},\top) = \begin{cases} 0 & \mathsf{T} = \top \\ 2 & \text{otherwise} \end{cases} \qquad d^{-}(\mathsf{B}_1,\mathsf{B}_2) = \begin{cases} 0 & \mathsf{B}_1 = \mathsf{B}_2 \\ 3 & \mathsf{B}_1 \neq \mathsf{B}_2 \end{cases}$$

$$d^{-}(\mathsf{T}_1 \to \mathsf{T}_2, \mathsf{T}'_1 \to \mathsf{T}'_2) = \begin{cases} d^{-}(\mathsf{T}_2,\mathsf{T}'_2) & \mathsf{T}_1 = \mathsf{T}'_1 \\ 3 & \mathsf{T}_1 \neq \mathsf{T}'_1 \end{cases}$$

$$d^{-}(\{\vec{l_1} : \vec{\mathsf{T}}_1\}, \{\vec{l_2} : \vec{\mathsf{T}}_2\}) = \begin{cases} 0 & \{\vec{l_1} : \vec{\mathsf{T}}_1\} = \{\vec{l_2} : \vec{\mathsf{T}}_2\} \\ 1 & \bar{l}_1 \subseteq \bar{l}_2 \text{ and } \max_{l \in \bar{l}_1} d^{-}(\mathsf{T}^l_1, \mathsf{T}^l_2) \leq 1 \\ 3 & \bar{l}_1 \cap \bar{l}_2 \neq \emptyset \text{ and } \max_{l \in \bar{l}_1 \cap \bar{l}_2} d^{-}(\mathsf{T}^l_1, \mathsf{T}^l_2) = 3 \\ 2 & \text{otherwise} \end{cases}$$

All other cases have a down-distance of 3. Benke [5] showed that the two following theorems hold.

Theorem 5.2.1 *Distance consistency is NLOGSPACE complete.*

Theorem 5.2.2 *A constraint is satisfiable if and only if it is weakly satisfiable and distance consistent.*

To highlight the separation of satisfiability into weak satisfiability and distance consistency we give two examples. The constraint $\mathsf{Int} \to \mathsf{X} <: \mathsf{Int} \to \mathsf{X} \to \mathsf{X}$ is distance consistent but not weakly satisfiable whereas the constraint $\mathsf{Int} <: \mathsf{Bool}$ is not distance consistent but weakly satisfiable.

Chapter 6

Conclusion

6.1 Related Work

Hindley/Milner type inference has been extended in numerous ways. For instance, Kfoury and Wells [17] consider inference for the rank-2 fragment of the second-order λ-calculus, later extended by Peyton Jones et al. [12] to arbitrary-rank types with the help of type annotations.

Hofmann and Pierce [11] introduce Positive Subtyping and enrich the traditional coercion interpretation of subtyping by adding an overwriting function. In addition to giving an equational theory for their calculus, they discuss differences between contra-variant subtyping and positive subtyping and give encodings of objects, interfaces and classes with self.

Aiken and Wimmers [2] consider type inclusion constraints and develop an algorithm to solve these constraints. In addition to the types we consider, they also include union types, intersection types, recursive types and a bottom type. However, they do not discuss complexity bounds. We believe that the algorithm they describe is applicable to the constraints that our type system produce, if the constraint simplification step were adapted to additionally constrain argument types to be positive. The type system we present is an instance of Sulzmann's HM(X) framework [36]. In comparison, we present an effective algorithmic version of type inference and give complexity bounds.

Palsberg [22] has shown that type inference for object types defined by Abadi and Cardelli [1] is PTIME-complete. In their object calculus, Abadi and Cardelli have width subtyping only and functional subtyping is invariant. Hoang and Mitchell [10] studied constraint satisfaction in presence of contra-variant subtyping and showed this problem can be PSPACE-hard.

Pierce and Turner's [27, 26] local type inference techniques, or their extension by Odersky et al. [21], focus on the same problem as our system - type inference in presence of subtyping - with a very different approach. These techniques are local and thus require some type annotations from the programmer, but remove the burden of writing silly types. Embedding local inference in a programming language is simpler and more flexible than Hindley/Milner type inference and has been chosen for the programming language Scala [20]. A common pitfall of local techniques is the difficulty for a programmer to understand which type annotations are necessary.

6.2 Future Work

Switching back to our work, we now outline future directions.

Verifying Type Annotations Allowing the programmer to give type annotations when interacting with the type system requires extra theory. These optional type annotations are given as type schemes, which must be typed in the polymorphic typing judgment. When type checking a program, the type system infers a type scheme, which ought to be checked against the type scheme provided by the programmer. Since our type system produces the most precise type (with respect to the subtyping relation on type schemes) for a term, we need to verify that the programmer's annotation is a supertype of the inferred type scheme. In our presentation, the subtyping relation on type schemes (definition 2.8.3) is semantic and its implementation is thus not obvious. We would need to give an efficient algorithmic presentation of the semantic subtyping relation on type schemes.

Recursive Types and Bottom Type The constraint system we have presented is syntactic and we therefore fail to type all pure terms. The Y-combinator for instance $\lambda x.(\lambda y.y\ (x\ x))\ (\lambda y.y\ (x\ x))$ produces a constraint that is not satisfiable.

Based on the work of Amadio and Cardelli [3], we would like to add recursive types to give a meaning to constraints of the form $\mathsf{X}_1 \rightarrow \mathsf{X}_2 <: \mathsf{X}_1$. It is nonetheless unclear how the efficiency of this extended system would compare to the one presented.

A more direct extension is the addition of a minimal type bottom ($\bot$), dual to the maximal type top ($\top$). The efficiency of our system would be unchanged and similar challenges to the one discussed by Pierce [25] are present in our system with top, making the introduction of this additional type simple.

Simplifying Types Constraint based type inference systems suffer from over-expressivity. Equivalent type schemes can take many shape and finding the "best" representation is non-trivial. In his work on bounded quantification with bottom, Pierce [25] alludes to similar difficulties in the syntactic representation of type schemes.

The type system we present does produce type schemes that are difficult to read. Constraints are aggregated and verified for satisfiability but never simplified. In the appendix, we have included type derivations of typical expressions and the unusual size of the inferred type is eye-catching. We finish these type derivations by using the declarative rule (BQ-DEC-SUB), when needed, to highlight the possibilities for simplification. A type scheme $\forall \bar{\mathsf{X}}' | C'.\mathsf{T}'$ is simpler than $\forall \bar{\mathsf{X}} | C.\mathsf{T}$ if it is equivalent but "syntactically less verbose". This definition is intentionally vague to give leeway for future work. We briefly discuss several ideas for simplifications, but leave as future work their formalization.

- Since we view constraints syntactically, and not as a set of subtyping inequalities, we must represent the empty constraint by true. Whenever constraints are conjuncted ((BQ-SD-REC), (BQ-SD-SEL), (BQ-SD-APP)) the true predicate could be eliminated instead of being carried forward.

- If a curried addition function is applied to a number, we would expect the type of the resulting expression to be $\mathsf{Int} \rightarrow \mathsf{Int}$ but our type system infers $\mathsf{S} = \forall \bar{\mathsf{X}} | \mathsf{Int} \rightarrow \mathsf{Int} \rightarrow \mathsf{Int} <: \mathsf{X}_1 \rightarrow \mathsf{X}_2 \wedge \mathsf{Int} <: \mathsf{X}_1 . \mathsf{X}_2$ whose useful solution is indeed $\mathsf{Int} \rightarrow \mathsf{Int}$. Here, the original type schemes' interpretation $[\![\mathsf{S}]\!]_\phi$ has a lower bound $\mathsf{Int} \rightarrow \mathsf{Int}$ which is as expressive as the type scheme itself. Whenever a lower bounded type scheme is inferred, the type inference system should use the lower bound instead of the more complex type scheme.

- Because of the way constraints are generated, extraneous type variables may be introduced. If the constraint $\mathsf{X}_1 <: \mathsf{X}_2 \wedge \mathsf{X}_2 <: \mathsf{X}_3$ appears in a type scheme whose monotype is $\mathsf{X}_1 \to \mathsf{X}_3$ then one could remove the intermediary X_2 type variable. Generally speaking, redundant constraints may be eliminated. Pottier and Rémy [29] give standard constraint equivalence laws which can be easily complemented with subtyping specific equivalences.

Aside from these typical difficulties with constraint-based typings, positive subtyping adds challenges of its own. Aiken and Wimmers [2] explain that in their type system, if a quantified type is monotonic (respectively anti-monotonic) in some type variable X, then X can be instantiated to the lower bound (respectively upper bound) implied by the constraint without altering the meaning of the type scheme. This simplification is not meaning preserving with positive subtyping but instead could reduce the precision of a type scheme. The type of the apply function discussed in the introduction exhibits this behaviour: the inferred type scheme is

$$\forall \bar{\mathsf{X}} | \mathsf{X}_1 <: \mathsf{X}_2 . \mathsf{X}_1 \to (\mathsf{X}_2 \to \mathsf{X}_3) \to \mathsf{X}_3$$

in which X_1 appears contra-variantly and X_2 appears co-variantly. We have seen previously that equating X_1 with X_2 reduces the expressiveness of the overall type scheme. An interesting counter-point is the other version of the apply function taking the function first and then the value. Its inferred type is

$$\forall \bar{\mathsf{X}} | \mathsf{X}_3 <: \mathsf{X}_1 . (\mathsf{X}_1 \to \mathsf{X}_2) \to \mathsf{X}_3 \to \mathsf{X}_2$$

which can be simplified to

$$\forall \bar{\mathsf{X}} | \mathsf{true} . (\mathsf{X}_1 \to \mathsf{X}_2) \to \mathsf{X}_1 \to \mathsf{X}_2$$

since after the first application, the resulting type will be $\forall \emptyset | \mathsf{true} . \mathsf{T}_1 \to \mathsf{T}_2$ for some types T_1 and T_2. The subtyping flexibility provided by the removed constraint is provided by the application rule. We conjecture that Aiken and Wimmers' optimization applies to type variables whose upper bound appears first.

To tackle this problem, we can regard curried functions' type as multi-instantiation types. Consider a curried function whose type is $\mathsf{T}_1 \to \mathsf{T}_2 \to \cdots \to \mathsf{T}_n$. When this function is first applied, the type system must decide the best choices for the type variables in T_1 (i.e. $\mathrm{ftv}(\mathsf{T}_1)$). In another portion of the type derivation, the resulting expression may be applied again and the type system must now find the best choice for type variables in T_2, and so on. For preciseness, when instantiating T_i's type variables, we must not restrict the range of $\mathsf{T}_{i+1} \to \cdots \to \mathsf{T}_n$. With positive subtyping, the types T_1 through T_{n-1} are invariant, and the constraint is there to recover the lost flexibility. When simplifying the constraint, we need to be certain to preserve enough flexibility.

Post-inference Variance Analysis Based on the observations about type simplification, it might be interesting to take a radically different approach. Instead of inferring a very general constrained type and simplifying it, we could infer a simple Hindley/Milner type and generalize it using a variance analysis.

The Hindley/Milner type of the apply function $\lambda x.\lambda f.f\ x$ is $\forall \mathsf{X}, \mathsf{Y}. \mathsf{X} \to (\mathsf{X} \to \mathsf{Y}) \to \mathsf{Y}$. With the standard subtyping relation, the first occurrence of the type variable X is anti-monotonic whereas the second occurrence is monotonic. Since these two occurrences conflict and they appear at different instantiation times, one must introduce a constraint to increase the expressiveness of the resulting type scheme.

Following this approach, one would have to fold subsumption into each typing rule to account for ground types. In the application rule for instance, if a function with type $\mathsf{T}_1 \to \mathsf{T}_2$ is applied to a grounded type T_1' we need to allow T_1' to be a subtype of T_1.

In the early steps of this research, we considered a "conformance" based type system using a post-inference variance analysis when typing expressions. The conformance relation is an superset of the positive subtyping relation but a strict subset of the standard contra-variant relation. This approach benefits from type conciseness, but its other advantages are unclear.

Satisfiability of Constraints Determining whether a constraint is satisfiable is key to the type inference's efficiency. We have show that constraint satisfiability is PTIME-complete and have presented a decomposition of this problem into weak satisfiability and distance consistency. We believe that a more direct algorithm tuned to the specific constraints generated by the type inference would be more effective in practice.

6.3 Conclusion

We have outlined the difficulties introduced by positive subtyping, a weaker notion of subtyping where contravariant positions are forced to be invariant. We have proposed a constraint based type inference system to recover the lost expressiveness and showed that, in our setting, satisfiability of the constraints generated by the type inference is PTIME-complete.

The programming language world is divided into two very separate communities, dynamically typed languages and statically typed languages. This divide goes back to Church and Curry, and has not weakened over the years.

In statically typed languages, inference relieves the programmer from writing silly types. Hindley/Milner type inference has set the standard, but is difficult to extend and quickly becomes intractable or undecidable. In complex languages such as Scala [20] local inference, pioneered by Pierce and Turner [27], is preferred. Until recently, mainstream programming languages did not benefit from these techniques. Despite the extensive background theory, introducing type inference and generics in Java [6, 15] has been difficult and shown buggy [16].

Statically typed languages are very well suited for large scale developments. The type system helps the programmer along the development cycle, allows for complex re-factoring and serves as a precise up-to-date documentation. In addition, statically typed languages can be aggressively optimized and thus very efficiently implemented.

On the other hand, dynamically typed languages benefit from a fast-paced compilation-less programming cycle making software development simpler and at the reach of beginners. It also makes software development very lightweight, appropriate for software whose malleability is crucial.

Making dynamically typed languages benefit from the research applicable to statically typed languages is a challenge of its own. Flanagan [8] introduced the concept of hybrid typing, later extended to gradual typing by Siek and Taha [34]. Rather than aiming at total type inference - undecidable for languages in which method override, removal and update is permitted - gradual typing partially infers types and leaves unknown types to be dynamically checked. This approach gives soft guarantees to programmers who can benefit from a fast paced prototyping phase and later add a few type annotations to crystallize their thinking, easing re-factoring and extendibility.

In this evolution, we see positive subtyping as one approach to tackle the complex structural subtyping relations induced by dynamically typed languages and thus improve the scope and preciseness of type inference in languages built for other purposes.

Appendix A

Typing Examples

We give a few examples from our prototype implementation[1] accessible on `http://www.stanford.edu/~plperez/`. The syntax used is close to the programming language ML. Constants comprises integers (e.g. `14`), booleans (`true` or `false`) and characters (e.g. `‘a’`). Our system infers type schemes which are typed in the polymorphic typing judgment. Our first example is the identity function:

```
fn x => x : \X1|true.X1 -> X1
```

The inferred type is the usual Hindley/Milner type. Our system prints type schemes as `\X|C.T` where `X` is the set of quantified type variables, `C` is the constraint and `T` is the monomorphic type. We simplify slightly the constraint for presentation. We now show an example of record selection, where the record type is extracted instead of adding a constraint.

```
#l {l=3} : \0|true.Int
```

In comparison, the selection function constrains the type variable representing the argument type.

```
fn x => #l x : \X1,X2|X1 <: {l:X2}.X1 -> X2
```

Our next example is the the Church encoding of a boolean $\lambda x.\lambda y.x$.

```
fn x => fn y => x : \X1,X2|true.X1 -> X2 -> X1
```

Its inferred type is similar to the Hindley/Milner type. In comparison with Aiken and Wimmers [2], the type variable `X2` cannot be instantiated to top as it appears in an invariant position. The apply function ($\lambda x.\lambda f.f\ x$) taking a value and then a function has the type discussed earlier, but the presentation is not as terse.

```
fn x => fn f => f x : \X1,X2,X3,X4|X1 <: X3 ^ X2 <: X3 -> X4.X1 -> X2 -> X4
```

To highlight let-polymorphism, we can apply the identity function to constants with different types.

```
let id = fn x => x in (fn w => id ‘a’) (id true) end :
    \X1,X2,X3|Bool <: X3 ^ X3 <: X1 ^ Char <: X2.X2
```

We use $(\lambda w.e_2)\ e_1$ as the consecution operator. Each individual constraint corresponds to one application and could be simplified when presenting the type scheme.

[1]The implementation is a work in progress and the distance consistency check is not yet incorporated.

Appendix B

Type Derivations Examples

B.1 Identity Function

$$\Delta_1 \quad \dfrac{\dfrac{\dfrac{\dfrac{(x : \forall\emptyset|\mathsf{true}.\mathsf{X})\,(x) = \forall\emptyset|\mathsf{true}.\mathsf{X}}{x : \forall\emptyset|\mathsf{true}.\mathsf{X} \vdash_{\mathsf{p}} x : \forall\emptyset|\mathsf{true}.\mathsf{X}}}{\mathsf{true}; x : \forall\emptyset|\mathsf{true}.\mathsf{X} \vdash_{\mathsf{m}} x : \mathsf{X}}}{\mathsf{true}; \emptyset \vdash_{\mathsf{m}} \lambda x.x : \mathsf{X} \to \mathsf{X}}}{\emptyset \vdash_{\mathsf{p}} \lambda x.x : \forall \mathsf{X}|\mathsf{true}.\mathsf{X} \to \mathsf{X}}$$

B.2 Identity Function applied to a Record

$$\dfrac{\dfrac{\dfrac{\dfrac{\Delta_1}{\mathsf{true}; \emptyset \vdash_{\mathsf{m}} \lambda x.x : \mathsf{X} \to \mathsf{X}} \qquad \dfrac{\dfrac{\dfrac{\mathscr{B}(4) = \mathsf{Int}}{\emptyset \vdash_{\mathsf{p}} 4 : \forall\emptyset|\mathsf{true}.\mathsf{Int}}}{\mathsf{true}; \emptyset \vdash_{\mathsf{m}} 4 : \mathsf{Int}}}{\mathsf{true}; \emptyset \vdash_{\mathsf{m}} \{l = 4\} : \{l : \mathsf{Int}\}}}{\mathsf{true} \wedge \mathsf{true} \wedge \mathsf{X} \to \mathsf{X} <: \mathsf{Y}_1 \to \mathsf{Y}_2 \wedge \{l : \mathsf{Int}\} <: \mathsf{Y}_1; \emptyset \vdash_{\mathsf{m}} \lambda x.x\ \{l = 4\} : \mathsf{Y}_2}}{\emptyset \vdash_{\mathsf{p}} \lambda x.x\ \{l = 4\} : \forall \{\mathsf{X}, \mathsf{Y}_1, \mathsf{Y}_2\}\,|\mathsf{true} \wedge \mathsf{true} \wedge \mathsf{X} \to \mathsf{X} <: \mathsf{Y}_1 \to \mathsf{Y}_2 \wedge \{l : \mathsf{Int}\} <: \mathsf{Y}_1.\mathsf{Y}_2}}{\emptyset \vdash_{\mathsf{p}} \lambda x.x\ \{l = 4\} : \forall\emptyset|\mathsf{true}.\,\{l : \mathsf{Int}\}}$$

B.3 Increment Function

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash_{\mathsf{m}} + : \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}} \qquad \dfrac{\vdots}{\vdash_{\mathsf{m}} x : \mathsf{X}_1}}{\overbrace{\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int} <: \mathsf{X}_2 \to \mathsf{X}_3 \wedge \mathsf{X}_1 <: \mathsf{X}_2}^{C_1}; x : \forall\emptyset|\mathsf{true}.\mathsf{X}_1 \vdash_{\mathsf{m}} x+ : \mathsf{X}_3} \qquad \dfrac{\vdots}{\vdash_{\mathsf{m}} 1 : \mathsf{Int}}}{\overbrace{C_1 \wedge \mathsf{X}_3 <: \mathsf{X}_4 \to \mathsf{X}_5 \wedge \mathsf{X}_3 <: \mathsf{X}_4}^{C_2}; x : \forall\emptyset|\mathsf{true}.\mathsf{X}_1 \vdash_{\mathsf{m}} x + 1 : \mathsf{X}_5}}{C_2; \emptyset \vdash_{\mathsf{m}} \lambda x.x + 1 : \mathsf{X}_1 \to \mathsf{X}_5}}{\emptyset \vdash_{\mathsf{p}} \lambda x.x + 1 : \forall\bar{\mathsf{X}}|C_2.\mathsf{X}_1 \to \mathsf{X}_5}}{\emptyset \vdash_{\mathsf{p}} \lambda x.x + 1 : \forall\emptyset|\mathsf{true}.\mathsf{Int} \to \mathsf{Int}}
$$

B.4 Selection Function

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{true}; x : \forall\emptyset|\mathsf{true}.\mathsf{X}_1 \vdash_{\mathsf{m}} x : \mathsf{X}_1}}{\mathsf{true} \wedge \mathsf{X}_1 <: \{l : \mathsf{X}_2\}; x : \forall\emptyset|\mathsf{true}.\mathsf{X}_1 \vdash_{\mathsf{m}} x.l : \mathsf{X}_2}}{\mathsf{true} \wedge \mathsf{X}_1 <: \{l : \mathsf{X}_2\}; \emptyset \vdash_{\mathsf{m}} \lambda x.x.l : \mathsf{X}_1 \to \mathsf{X}_2}}{\emptyset \vdash_{\mathsf{p}} \lambda x.x.l : \forall\bar{\mathsf{X}}|\mathsf{true} \wedge \mathsf{X}_1 <: \{l : \mathsf{X}_2\}.\mathsf{X}_1 \to \mathsf{X}_2}}{\emptyset \vdash_{\mathsf{p}} \lambda x.x.l : \forall\mathsf{X}|\mathsf{true}.\{l : \mathsf{X}\} \to \mathsf{X}}
$$

B.5 Apply Function (f then d)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{true}; \Gamma \vdash_{\mathsf{m}} f : \mathsf{X}_1} \qquad \dfrac{\vdots}{\mathsf{true}; \Gamma \vdash_{\mathsf{m}} d : \mathsf{X}_2}}{\overbrace{\mathsf{X}_1 <: \mathsf{X}_3 \to \mathsf{X}_4 \wedge \mathsf{X}_2 <: \mathsf{X}_3}^{C}; \overbrace{f : \forall\emptyset|\mathsf{true}.\mathsf{X}_1 \cdot d : \forall\emptyset|\mathsf{true}.\mathsf{X}_2}^{\Gamma} \vdash_{\mathsf{m}} f\ d : \mathsf{X}_4}}{C; f : \forall\emptyset|\mathsf{true}.\mathsf{X}_1 \vdash_{\mathsf{m}} \lambda d.f\ d : \mathsf{X}_2 \to \mathsf{X}_4}}{C; \emptyset \vdash_{\mathsf{m}} \lambda f.\lambda d.f\ d : \mathsf{X}_1 \to \mathsf{X}_2 \to \mathsf{X}_4}}{\dfrac{\emptyset \vdash_{\mathsf{p}} \lambda f.\lambda d.f\ d : \forall\bar{\mathsf{X}}|C.\mathsf{X}_1 \to \mathsf{X}_2 \to \mathsf{X}_4}{\emptyset \vdash_{\mathsf{p}} \lambda f.\lambda d.f\ d : \forall\bar{\mathsf{X}}|\mathsf{true}.(\mathsf{X}_1 \to \mathsf{X}_2) \to \mathsf{X}_1 \to \mathsf{X}_2}}
$$

B.6 Apply Function (d then f)

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{true}; \Gamma \vdash_{\mathsf{m}} f : \mathsf{X}_2} \qquad \dfrac{\vdots}{\mathsf{true}; \Gamma \vdash_{\mathsf{m}} d : \mathsf{X}_1}}{\overbrace{\mathsf{X}_2 <: \mathsf{X}_3 \to \mathsf{X}_4 \wedge \mathsf{X}_1 <: \mathsf{X}_3}^{C}; \overbrace{d : \forall\emptyset|\mathsf{true}.\mathsf{X}_1 \cdot f : \forall\emptyset|\mathsf{true}.\mathsf{X}_2}^{\Gamma} \vdash_{\mathsf{m}} f\ d : \mathsf{X}_4}}{C; d : \forall\emptyset|\mathsf{true}.\mathsf{X}_1 \vdash_{\mathsf{m}} \lambda f.f\ d : \mathsf{X}_1 \to \mathsf{X}_4}}{C; \emptyset \vdash_{\mathsf{m}} \lambda d.\lambda f.f\ d : \mathsf{X}_2 \to \mathsf{X}_1 \to \mathsf{X}_4}}{\emptyset \vdash_{\mathsf{p}} \lambda d.\lambda f.f\ d : \forall\bar{\mathsf{X}}|C.\mathsf{X}_2 \to \mathsf{X}_1 \to \mathsf{X}_4}}{\emptyset \vdash_{\mathsf{p}} \lambda d.\lambda f.f\ d : \forall\bar{\mathsf{X}}|\mathsf{X}_1 <: \mathsf{X}_2.\mathsf{X}_1 \to (\mathsf{X}_2 \to \mathsf{X}_3) \to \mathsf{X}_3}
$$

Appendix C

Notations

c	constant	7
x, y	variables	7
$\top$	top type	7
B	base type	7
X, Y	type variables	7
T, U	monomorphic types	7
S	type schemes	8
$\mathsf{T}_1 <: \mathsf{T}_2$	monomorphic subtyping relation	9
$\mathsf{S}_1 <: \mathsf{S}_2$	polymorphic subtyping relation	14
l	record label	7
$\mathcal{L}$	set of labels	7
$\mathcal{V}$	set of type variables	7
$\mathcal{M}$	set of ground types	8
$\mathcal{T}$	set of monomorphic types	7
$\mathcal{S}$	set of type schemes	8
$\mathcal{B}$	mapping of constant to type schemes	9
Γ, Δ	typing environment	9
fv	free variables of an expression	7
ftv	free type variables	8
C, D	constraints	8
$\uparrow$	upward closure	11
$\downarrow$	downward closure	11
ψ	map	11
ϕ	ground assignement	11
$\theta_{\bar{\mathsf{X}}}$	substitution of the type variables in $\bar{\mathsf{X}}$ to types or ground types	12
$\phi \models C$	constraint satisfaction	12

Bibliography

[1] Martín Abadi and Luca Cardelli. A theory of primitive objects: Second-order systems. In Donald Sannella, editor, *Proceeding of ESOP '94 on Programming Languages and Systems*, volume 788, pages 1–25. Springer Verlag, 1994.

[2] Alexander Aiken and Edward L. Wimmers. Type inclusion constraints and type inference. In *FPCA '93: Proceedings of the conference on Functional programming languages and computer architecture*, pages 31–41, New York, NY, USA, 1993. ACM Press.

[3] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, September 1993.

[4] Christopher Anderson, Paola Giannini, and Sophia Drossopoulou. Towards type inference for javascript. In *ECOOP*, volume 3586, 2005.

[5] Marcin Benke. Efficient type reconstruction in the presence of inheritance. In *MFCS*, pages 272–280, 1993.

[6] Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the future safe for the past: Adding genericity to the Java programming language. In Craig Chambers, editor, *ACM Symposium on Object Oriented Programming: Systems, Languages, and Applications (OOPSLA)*, pages 183–200, Vancouver, BC, 1998.

[7] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *POPL '82: Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 207–212, New York, NY, USA, 1982. ACM Press.

[8] Cormac Flanagan. Hybrid type checking. In *POPL '06: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 245–256, New York, NY, USA, 2006. ACM Press.

[9] My K. Hoang. *Type Inference and Program Evaluation in the Presence of Subtyping*. PhD thesis, Stanford University, 1995.

[10] My K. Hoang and John C. Mitchell. Lower bounds on type inference with subtypes. In *POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 176–185, New York, NY, USA, 1995. ACM Press.

[11] Martin Hofmann and Benjamin Pierce. Positive subtyping. In *POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 186–197, New York, NY, USA, 1995. ACM Press.

[12] S. Peyton Jones, D. Vytiniotis, S. Weirich, and M. Shields. Practical type inference for arbitrary-rank types. *Under consideration for publication in J. Functional Programming*, 2006.

[13] Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Geoffrey Washburn. Simple unification-based type inference for gadts. *SIGPLAN Not.*, 41(9):50–61, 2006.

[14] P.C. Kanellakis, H.G. Mairson, and J.C. Mitchell. Unification and ML-Type Reconstruction. *Computational Logic-Essays in Honor of Alan Robinson*, pages 444–478, 1991.

[15] David Holmes Ken Arnold, James Gosling. *Java^{TM}Programming Language, 4th Edition*. Prentice Hall PTR, 2005.

[16] Andrew J. Kennedy and Benjamin C. Pierce. On decidability of nominal subtyping with variance, September 2006. FOOL-WOOD '07.

[17] 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 *LFP '94: Proceedings of the 1994 ACM conference on LISP and functional programming*, pages 196–207, New York, NY, USA, 1994. ACM Press.

[18] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[19] Peter Nevermann and Ivan Rival. Holes in ordered sets. *Graphs and Combinatorics*, 1:339–350, 1985.

[20] Martin Odersky and al. An overview of the scala programming language. Technical Report IC/2004/64, EPFL Lausanne, Switzerland, 2004.

[21] Martin Odersky, Matthias Zenger, and Christoph Zenger. Colored local type inference. In *Proc. ACM Symposium on Principles of Programming Languages*, pages 41–53, 2001.

[22] Jens Palsberg. Efficient inference of object types. In *Logic in Computer Science*, pages 186–195, 1994.

[23] Emir Pasalic, Jeremy G. Siek, and Walid Taha. Concoqtion: Mixing indexed types and hindley-milner type inference. In *POPL '07: Conference record of the 34th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2007. submitted.

[24] M. S. Paterson and M. N. Wegman. Linear unification. In *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 181–186, New York, NY, USA, 1976. ACM Press.

[25] Benjamin C. Pierce. Bounded quantification with bottom. Technical Report 492, Computer Science Department, Indiana University, 1997.

[26] Benjamin C. Pierce and David N. Turner. Local type argument synthesis with bounded quantification. Technical Report 495, Computer Science Department, Indiana University, January 1997.

[27] Benjamin C. Pierce and David N. Turner. Local type inference. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), San Diego, California*, 1998. Full version in *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22(1), January 2000, pp. 1–44.

[28] François Pottier. A constraint-based presentation and generalization of rows. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 331, Washington, DC, USA, 2003. IEEE Computer Society.

[29] François Pottier and Didier Rémy. The essence of ML type inference. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. MIT Press, 2005.

[30] Vaughan R. Pratt and Jerzy Tiuryn. Satisfiability of inequalities in a poset. *Fundamenta Informaticae*, 28(1-2):165–182, 1996.

[31] Alain Quilliot. An application of the helly property to the partially ordered sets. *Journal of Combinatorial Theory*, 35:185–198, 1983.

[32] Didier Rémy. Projective ML. In *1992 ACM Conference on Lisp and Functional Programming*, pages 66–75, New-York, 1992. ACM press.

[33] Didier Rémy. Type inference for records in a natural extension of ML. In Carl A. Gunter and John C. Mitchell, editors, *Theoretical Aspects Of Object-Oriented Programming. Types, Semantics and Language Design*. MIT Press, 1993.

[34] Jeremy Siek and Walid Taha. Gradual typing for objects. Sent to the `[TYPES]` list, available on http://www.cs.colorado.edu/~siek/gradual-obj.pdf, February 2007.

[35] Martin Sulzmann. *A General Framework for Hindley/Milner Type Systems with Constraints*. PhD thesis, Yale University, Department of Computer Science, 2000.

[36] Martin Sulzmann, Martin Odersky, and Martin Wehr. Type inference with constrained types. In *Fourth International Workshop on Foundations of Object-Oriented Programming (FOOL 4)*, 1997.

[37] Jerzy Tiuryn. Subtype inequalities. In *LICS*, pages 308–315, 1992.

[38] J. Wells. The undecidability of mitchell's subtyping relation. Technical report, Boston University, 1995.