Rank 2 type systems and recursive definitions

Technical Memorandum MIT/LCS/TM–531

Trevor Jim*
Laboratory for Computer Science
Massachusetts Institute of Technology

August 1995; revised November 1995

Abstract

We demonstrate an equivalence between the rank 2 fragments of the polymorphic lambda calculus (System F) and the intersection type discipline: exactly the same terms are typable in each system. An immediate consequence is that typability in the rank 2 intersection system is DEXPTIME-complete. We introduce a rank 2 system combining intersections and polymorphism, and prove that it types exactly the same terms as the other rank 2 systems. The combined system suggests a new rule for typing recursive definitions. The result is a rank 2 type system with decidable type inference that can type some interesting examples of polymorphic recursion. Finally, we discuss some applications of the type system in data representation optimizations such as unboxing and overloading.

Keywords: Rank 2 types, intersection types, polymorphic recursion, boxing/unboxing, overloading.

1 Introduction

In the past decade, Milner's type inference algorithm for ML has become phenomenally successful. As the basis of popular programming languages like Standard ML and Haskell, Milner's algorithm is the preferred method of type inference among language implementors. And in the theoretical

*545 Technology Square, Cambridge, MA 02139, trevor@theory.lcs.mit.edu. Supported by NSF grants CCR–9113196 and CCR–9417382, and ONR Contract N00014–92–J–1310.

community, the literature on type inference is dominated by extensions of ML's let-polymorphism.

In this paper we examine some alternatives to ML that have attracted surprisingly little attention: the systems of rank 2 types introduced by Leivant [21]. These systems are slightly more powerful than ML—strictly more terms can be assigned types—and the increased power comes for free—the complexity of typability is identical. But the unique feature of the rank 2 systems that justifies further study is that, in sharp contrast to other extensions of ML, they abandon let-polymorphism.

We use the expression $(\lambda x.xx)$ to illustrate the limitations of let-polymorphism. It is well known that this expression cannot be typed in ML: the only way for ML to type the self-application xx is by assigning a polymorphic type to x, and ML does not allow abstraction over variables with polymorphic type. In ML, the only mechanism for introducing variables of polymorphic type is the let-expression:

$$\begin{array}{l}\textbf{let } x = (\lambda y.y) \\ \textbf{in } xx\end{array}$$

This let-expression binds x to the identity function $(\lambda y.y)$, which has the polymorphic type $\forall t.t \to t$ in ML. By ML's let-polymorphism, x is assigned the type $\forall t.t \to t$, which is sufficient to type xx.

The problem with this is that we cannot typecheck the uses of x (the application xx) separately from its definition (the function $(\lambda y.y)$). So ML must be extended with a *module language* in order to support programming in the large, where it is impractical to require every polymorphic definition to appear in the same source file as every use.

In contrast, $(\lambda x.xx)$ is typable in all of the rank 2 systems we consider. Here are two rank 2 typings:

$$(\lambda x.xx) : (\forall t.t \to t) \to (\forall s.s \to s),$$

$$(\lambda x.xx) : (t \to t) \wedge ((t \to t) \to (t \to t)) \to (t \to t).$$

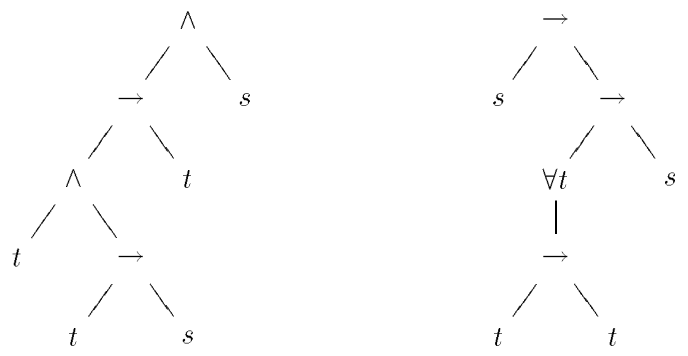
The first typing says that $(\lambda x.xx)$ is a function that, when given an argument with type $t \to t$ for *any* type t, produces a result with type $s \to s$, for any s. The identity function is an appropriate argument.

The second typing says that $(\lambda x.xx)$ is a function that, when given an argument having *both* the types $(t \to t)$ and $(t \to t) \to (t \to t)$, produces a result of type $(t \to t)$. Once again, the identity $(\lambda y.y)$ is an appropriate argument.

The rank 2 systems we consider are subsystems of two widely studied type systems, System F and the system of intersection types. System F, introduced independently by Girard [7] and by Reynolds [28], predates ML and can type many more terms. A recent result of Wells [34], however, shows that typability in the system is undecidable, putting type inference out of reach.

The system of intersection types, introduced independently by Coppo and Dezani [5] and by Sallé [29], can type even more terms than System F: it types all (and only) the strongly normalizing terms.[1] The equivalence of typability and strong normalization implies that type inference, just as with System F, is unattainable.

With the goal of type inference in mind, we seek decidable restrictions of these type systems. Restrictions based on the *rank* of types were suggested by Leivant [21]. The rank of a type can be easily determined by examining it in tree form. A type is of rank k if no path from the root of the type to a type constructor of interest (either type intersection '$\wedge$' or type quantification '$\forall$') passes to the left of k arrows. The types shown in Figure 1 are rank 2 types, because no path from root to $\wedge$ or $\forall$ passes to the left of two arrows. But the types shown in Figure 2 go beyond rank 2 (they are rank 3 types). The types given above for $(\lambda x.xx)$ are rank 2 types.

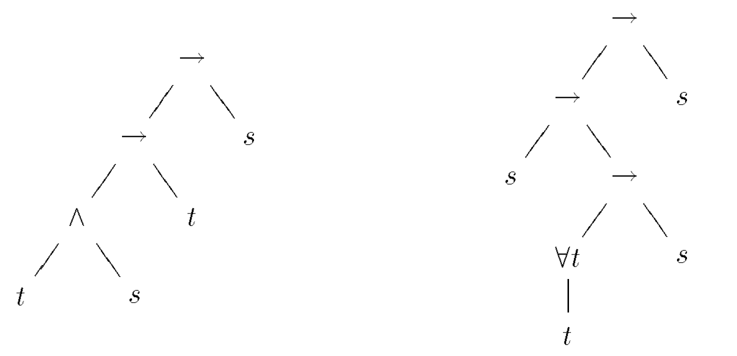


$((t \wedge (t \rightarrow s)) \rightarrow t) \wedge s$ $\qquad$ $s \rightarrow (\forall t.t \rightarrow t) \rightarrow s$

Figure 1: Examples of rank 2 types

Ranks 0 and 1 of Leivant's systems are equivalent to the simply typed

[1]Without the type constant ω.



$((t \wedge s) \to t) \to s$ $\qquad\qquad$ $(s \to (\forall t.t) \to s) \to s$

Figure 2: Types that go beyond rank 2

lambda calculus, which can type fewer terms than ML. But starting with rank 2, the systems can type more terms than ML.

Rank 2 of System F, which we call Λ_2, has received the most study. McCracken [23] proposed a type inference algorithm for Λ_2 based on Leivant's ideas. This algorithm is incorrect. Kfoury and Tiuryn [12] show that the complexity of typability in Λ_2 is identical to that of ML. Kfoury and Wells [16, 17] give a correct type inference algorithm, and show that ranks 3 and higher in System F are undecidable.

Leivant's original paper is almost the only work on rank 2 of the intersection type discipline, which we call $\mathbf{I}_2$. Leivant sketched a type inference algorithm for $\mathbf{I}_2$, but the algorithm was not formalized and proved correct until recently [33]. Leivant also conjectured the undecidability of ranks 3 and higher in the intersection system; to our knowledge the details of his proof idea have never been verified.

$\mathbf{I}_2$ has a significant advantage over Λ_2: it has *principal typings.* This means that for any term M, if M is typable in $\mathbf{I}_2$, then there is an $\mathbf{I}_2$ typing judgment

$$A \vdash M : \sigma$$

that represents all of the possible typing judgments for M. Other typings for M can be obtained from the principal typing by simple operations (substitution and subsumption).

Contributions of the paper

Since $\mathbf{I}_2$ has principal typings, and Λ_2 does not, we believe $\mathbf{I}_2$ deserves more study. The first contribution of this paper is to develop some of the basic properties of $\mathbf{I}_2$. We establish the following equivalence:[2] a term is typable in $\mathbf{I}_2$ if and only if it is typable in Λ_2. An immediate corollary is that typability in $\mathbf{I}_2$ is DEXPTIME-complete, identical to typability in Λ_2 and ML. We also consider some variants of $\mathbf{I}_2$, and show they are all equivalent in terms of typability.

The second contribution of this paper is to introduce a new type system, $\mathbf{P}_2$, that combines rank 2 intersection types and top-level quantification of type variables, as in ML. $\mathbf{P}_2$ has principal typings, so it clearly improves on Λ_2. Its advantage over $\mathbf{I}_2$ is more subtle. The addition of quantifiers makes types more expressive: the quantifiers identify *generic* type variables, that is, type variables which can safely be instantiated with any type. In particular, this suggests an interesting type inference algorithm for *recursive definitions*.

A recursive definition is written in the form $(\mu x M)$, and is meant to denote a program x such that

$$x \quad = \quad M,$$

where M may contain some uses of x. The standard rule for typing recursive definitions looks like

$$\frac{A \cup \{x : \sigma\} \vdash M : \sigma}{A \vdash (\mu x M) : \sigma}$$

Most type inference algorithms restrict the type σ in this rule to be a simple type. The rule of *polymorphic recursion* relaxes this restriction by allowing σ to be an ML type scheme. This gives a useful increase in typing power—it can type some natural programs that cannot be typed by the simple recursion rule. However, polymorphic recursion makes type inference undecidable [14].

We suggest another way of typing recursive definitions:

$$\frac{A \cup \{x : \tau\} \vdash M : \sigma}{A \vdash (\mu x M) : \sigma} \quad (\text{where } \sigma \leq \tau)$$

The rule says that as long as the type σ of M is more general than the assumption τ on x needed to type M, we can deduce σ as the type of the recursive definition.

[2]The equivalence between the rank 2 fragments of System F and the intersection type discipline has been shown independently by Yokouchi [35].

We extend $\mathbf{P}_2$ to type recursive definitions in this way. The resulting system can type many (but not all) of the examples that seem to require polymorphic recursion. Moreover, the system has principal types and decidable type inference.

Organization of the paper

In §2, we introduce $\mathbf{I}_2^{\mathrm{s}}$, a syntax-directed version of $\mathbf{I}_2$, and Λ_2^{s}, a syntax-directed version of Λ_2. The main result is that a term is typable in one system if and only if it is typable in the other. An immediate corollary is that typability in $\mathbf{I}_2^{\mathrm{s}}$ is DEXPTIME-complete, the same complexity as in ML and Λ_2^{s}. In §3, we present the type inference algorithm for $\mathbf{I}_2^{\mathrm{s}}$. In §4, we discuss some other definitions of rank 2 intersection type systems, and show their equivalence with $\mathbf{I}_2$. In §5, we define $\mathbf{P}_2$, show that it has principal typings, and give a type inference algorithm. In §6, we discuss various ways of typing recursive definitions, and we propose an extension of $\mathbf{P}_2$ that can type many examples of polymorphic recursion. We discuss applications of $\mathbf{P}_2$ to compilation in §7, and we summarize our results in §8.

2 Rank 2 type systems

2.1 Preliminaries

We will be defining a number of type systems; here we develop machinery that will be useful in all of them.

We use $x, y, \ldots$ to range over a countable set of variables, and t, s to range over a countable set, $\mathbf{Tv}$, of type variables. The terms and types of the systems will vary, but in all cases we use $\sigma, \tau, \ldots$ to range over types, and M, N, ... to range over terms.

The *terms of the (pure) lambda calculus* are defined by the following grammar:

$$M \quad ::= \quad x \mid (M_1 M_2) \mid (\lambda x M).$$

Unless stated otherwise, terms are considered syntactically equal modulo renaming of bound variables. We adopt the usual conventions that allow us to omit parentheses: application associates to the left, and the scope of an abstraction 'λ' extends to the right as far as possible. We write $\lambda x_1 \cdots x_n.M$ for $(\lambda x_1(\cdots(\lambda x_n M)\cdots))$.

The types of our systems will all be subsets of the types with quantification and intersection:

$$\sigma \quad ::= \quad t \mid (\sigma_1 \to \sigma_2) \mid (\forall t \sigma) \mid (\sigma_1 \wedge \sigma_2).$$

By convention, '$\rightarrow$' associates to the right, so that, e.g., $(t \rightarrow (t \rightarrow t))$ may be written more compactly as $t \rightarrow t \rightarrow t$, and '$\wedge$' binds more tightly than '$\rightarrow$', e.g., $\sigma \wedge \tau \rightarrow t$ means $(\sigma \wedge \tau) \rightarrow t$. The scope of a quantifier '$\forall$' extends as far to the right as possible. We write $(\forall \vec{t} \sigma)$ for the type

$$(\forall t_1 (\forall t_2 (\ldots (\forall t_n \sigma) \ldots))),$$

where $\vec{t} = t_1, t_2, \ldots, t_n$ and $n \geq 0$.

The set of *simple types*, $\mathbf{T}_0$, is defined by the following inductive equation:

$$\mathbf{T}_0 \quad = \quad \{\, t \mid t \text{ is a type variable} \,\} \cup \{\, (\sigma \rightarrow \tau) \mid \sigma, \tau \in \mathbf{T}_0 \,\}.$$

A *type environment* is a finite set $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ of (variable, type) pairs, where the variables $x_1, \ldots, x_n$ are distinct. We use A to range over type environments. We write $A(x)$ for the type paired with x in A, $\mathbf{dom}(A)$ for the set $\{x \mid \exists \tau.(x : \tau) \in A\}$, and A_x for the type environment A with any pair for the variable x removed. We write $A_1 \cup A_2$ for the union of two type environments; by convention we assume that $\mathbf{dom}(A_1)$ and $\mathbf{dom}(A_2)$ are disjoint. For any set $\mathbf{T}$ of types, we say A is a $\mathbf{T}$ type environment if $A(x) \in \mathbf{T}$ for all $x \in \mathbf{dom}(A)$.

The notion of *free type variable* is defined as usual. We write $\mathrm{FTV}(\sigma)$ for the free type variables of a type σ, and $\mathrm{FTV}(A)$ for the free type variables of all types appearing in A. We write $\mathrm{Gen}(A, \tau)$ for the $\forall$-closure of τ by the type variables $\mathrm{FTV}(\tau) - \mathrm{FTV}(A)$.

A *judgment* is a relation between type environments, terms, and types, written $A \vdash M : \sigma$. A term M is *typable* if $A \vdash M : \sigma$ for some A and σ. A pair $\langle A, \sigma \rangle$ of a type environment and a type is called simply a *pair*. Two pairs $\langle A_1, \sigma_1 \rangle$ and $\langle A_2, \sigma_2 \rangle$ are *disjoint* if their free type variables are disjoint. An *acceptable pair of a term M* in a type system is a pair $\langle A, \sigma \rangle$ such that the judgment $A \vdash M : \sigma$ holds in the type system. We write $\mathrm{AP}_{\mathcal{T}}(M)$ for the set of acceptable pairs of M in a type system $\mathcal{T}$.

A *substitution* is a mapping from type variables to simple types which is the identity on all but a finite number of type variables. We use S, R, Q, U to range over substitutions. The *domain* and *range* of a substitution S are defined

$$\begin{aligned} \mathbf{dom}(S) &= \{t \mid St \neq t\}, \\ \mathbf{rng}(S) &= \bigcup_{t \in \mathbf{dom}(S)} \mathrm{FTV}(St). \end{aligned}$$

If $\mathbf{dom}(S) = \{t_1, t_2, \ldots, t_n\}$ and $St_i = \tau_i$ for all i, then S can be written in the form $\{t_1 := \tau_1, \ldots, t_n := \tau_n\}$.

The application of substitutions is extended to types, type environments, and pairs in the usual way. The composition of substitutions is denoted by juxtaposition, so that $SRt = (SR)t = S(R(t))$. We say S_1 and S_2 are *disjoint* if $\mathbf{dom}(S_1)$ and $\mathbf{dom}(S_2)$ are disjoint sets. If S_1 and S_2 are disjoint, then the substitution $S_1 \cup S_2$ is defined as follows:

$$(S_1 \cup S_2)(t) = \begin{cases} S_1(t) & \text{if } t \in \mathbf{dom}(S_1), \\ S_2(t) & \text{if } t \in \mathbf{dom}(S_2), \\ t & \text{otherwise.} \end{cases}$$

Note that we have made a severe restriction on substitutions: they map type variables only to simple types, and not types in general.

2.2 The rank 2 intersection type system

There are many different formulations of intersection type systems; see van Bakel [33] for a survey. We will present a very restricted intersection type system here, the system of rank 2 intersection types. Our system is a slight generalization of van Bakel's version (see §4.1).

The *terms of the intersection type system* are just the terms of the lambda calculus. The sets $\mathbf{T}_1$ and $\mathbf{T}_2$ are defined to be the smallest sets satisfying the following equations:

$$\begin{aligned} \mathbf{T}_1 &= \mathbf{T}_0 \cup \{(\sigma \wedge \tau) \mid \sigma, \tau \in \mathbf{T}_1\}, \\ \mathbf{T}_2 &= \mathbf{T}_0 \cup \{(\sigma \rightarrow \tau) \mid \sigma \in \mathbf{T}_1, \tau \in \mathbf{T}_2\}. \end{aligned}$$

The set $\mathbf{T}_1$ of rank 1 types consists of finite, nonempty intersections of simple types. $\mathbf{T}_2$ is the set of rank 2 intersection types: these are types possibly containing intersections, but only to the left of a single arrow. Note that $\mathbf{T}_0 = \mathbf{T}_1 \cap \mathbf{T}_2$, and for $i \in \{0, 1, 2\}$, if $\tau \in \mathbf{T}_i$, then $S\tau \in \mathbf{T}_i$.

In order to simplify subsequent definitions, we adopt the following syntactic convention: we consider '$\wedge$' to be an associative, commutative, and idempotent operator, so that any $\mathbf{T}_1$ type may be considered a finite, nonempty set of simple types, written in the form $(\bigwedge_{i \in I} \sigma_i)$, where each $\sigma_i \in \mathbf{T}_0$.

Definition 1 For $i \in \{1, 2\}$, we define the relation $\leq_i$ as the least partial order on $\mathbf{T}_i$ closed under the following rules:

i) If $\{\tau_j \mid j \in J\} \subseteq \{\sigma_i \mid i \in I\}$, then $(\bigwedge_{i \in I} \sigma_i) \leq_1 (\bigwedge_{j \in J} \tau_j)$.

$$(\text{VAR}) \quad A \cup \{x : (\bigwedge_{i\in I} \tau_i)\} \vdash x : \tau_{i_0} \quad (\text{where } i_0 \in I)$$

$$(\text{ABS}) \quad \frac{A_x \cup \{x : \sigma\} \vdash M : \tau}{A \vdash (\lambda x M) : \sigma \to \tau}$$

$$(\text{APP}) \quad \frac{A \vdash M : (\bigwedge_{i\in I} \tau_i) \to \sigma, \quad (\forall i \in I)\ A \vdash N : \tau_i}{A \vdash (MN) : \sigma}$$

Figure 3: Typing rules of $\mathbf{I}_2^{\mathrm{s}}$. Types in type environments are in $\mathbf{T}_1$, and derived types are in $\mathbf{T}_2$.

ii) If $\sigma_1 \leq_1 \tau_1$ and $\tau_2 \leq_2 \sigma_2$, then $(\tau_1 \to \tau_2) \leq_2 (\sigma_1 \to \sigma_2)$.

The first rule says that $\leq_1$ expresses the natural ordering on intersection types, and the second rule says that $\leq_2$ obeys the usual antimonotonic ordering on function types, restricted to rank 2.

Some useful properties of the orderings $\leq_1$ and $\leq_2$ are summarized in the following lemma.

Lemma 2

i) *If $\sigma \in \mathbf{T}_0$ and $\tau \in \mathbf{T}_1$, then $\sigma \leq_1 \tau$ iff $\sigma = \tau$.*

ii) *If $\sigma \in \mathbf{T}_2$ and $\tau \in \mathbf{T}_0$, then $\sigma \leq_2 \tau$ iff $\sigma = \tau$.*

iii) *For $i \in \{1, 2\}$, if $\sigma \leq_i \tau$, then $S\sigma \leq_i S\tau$.*

Judgments in our rank 2 system are defined inductively by the rules of Figure 3. We write $\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$ if the judgment $A \vdash M : \sigma$ follows by these rules, with types appearing in type environments restricted to $\mathbf{T}_1$, and derived types restricted to $\mathbf{T}_2$. The superscript 's' in $\mathbf{I}_2^{\mathrm{s}}$ indicates that the system is syntax-directed, in contrast with a later variant (see §4).

If A_1 and A_2 are $\mathbf{T}_1$ type environments, we define $A_1 + A_2$, a $\mathbf{T}_1$ type environment, as follows: for each $x \in \mathbf{dom}(A_1) \cup \mathbf{dom}(A_2)$,

$$(A_1 + A_2)(x) = \begin{cases} A_1(x) & \text{if } x \notin \mathbf{dom}(A_2), \\ A_2(x) & \text{if } x \notin \mathbf{dom}(A_1), \\ A_1(x) \wedge A_2(x) & \text{otherwise.} \end{cases}$$

Lemma 3 (Weakening) *If $\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$, then $\mathbf{I}_2^{\mathrm{s}} \triangleright A + A' \vdash M : \sigma$ for any $\mathbf{T}_1$ type environment A'.*

Proof: An easy induction on typing derivations. $\square$

Lemma 4 (Substitutivity) *If* $\mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M : \sigma$*, then* $\mathbf{I}_2^{\mathrm{s}} \rhd SA \vdash M : S\sigma$ *for any substitution* S*.*

Proof: By induction on the structure of M.

i) If $M = x$, then $A(x) = (\bigwedge_{i \in I} \sigma_i)$ and $\sigma = \sigma_{i_0}$ for some $i_0 \in I$. Then $SA(x) = (\bigwedge_{i \in I} S\sigma_i)$, $\mathbf{I}_2^{\mathrm{s}} \rhd SA \vdash x : S\sigma_{i_0}$, and $S\sigma = S\sigma_{i_0}$.

ii) If $M = \lambda x N$ then σ must be of the form $\tau_1 \to \tau_2$, and $\mathbf{I}_2^{\mathrm{s}} \rhd A_x \cup \{x : \tau_1\} \vdash N : \tau_2$. Then by induction, $\mathbf{I}_2^{\mathrm{s}} \rhd S(A_x \cup \{x : \tau_1\}) \vdash N : S\tau_2$, so by rule (ABS), $\mathbf{I}_2^{\mathrm{s}} \rhd SA_x \vdash N : S\tau_1 \to S\tau_2$, or $\mathbf{I}_2^{\mathrm{s}} \rhd SA_x \vdash N : S(\tau_1 \to \tau_2)$. Then by weakening, $\mathbf{I}_2^{\mathrm{s}} \rhd SA \vdash N : S(\tau_1 \to \tau_2)$.

iii) If $M = M_1 M_2$, then for some $(\bigwedge_{i \in I} \tau_i) \in \mathbf{T}_1$ we have $\mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M_1 : (\bigwedge_{i \in I} \tau_i) \to \sigma$ and $\mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M_2 : \tau_i$ for all $i \in I$. By induction we have $\mathbf{I}_2^{\mathrm{s}} \rhd SA \vdash M_1 : (\bigwedge_{i \in I} S\tau_i) \to S\sigma$ and $\mathbf{I}_2^{\mathrm{s}} \rhd SA \vdash M_2 : S\tau_i$, and by rule (APP), we have $\mathbf{I}_2^{\mathrm{s}} \rhd SA \vdash M_1 M_2 : S\sigma$, as desired.

$\square$

2.3 System F

The *terms of System F* are exactly the terms of the lambda calculus. The *types of System F* are defined by the following grammar:

$$\tau \quad ::= \quad t \mid (\tau_1 \to \tau_2) \mid (\forall t \tau).$$

We consider System F types to be syntactically equal modulo renaming of bound type variables, reordering of adjacent quantifiers, and elimination of unnecessary quantifiers.

The types of System F can be organized into a hierarchy as follows. First, define $\mathbf{R}(0) = \mathbf{T}_0$. Then for $n \geq 0$, the set $\mathbf{R}(n+1)$ is defined to be the least set satisfying

$$\begin{aligned} \mathbf{R}(n+1) \quad &= \quad \mathbf{R}(n) \cup \{(\sigma \to \tau) \mid \sigma \in \mathbf{R}(n), \tau \in \mathbf{R}(n+1)\} \\ &\qquad \cup \{(\forall t \sigma) \mid \sigma \in \mathbf{R}(n+1)\}. \end{aligned}$$

It will be useful to restrict types so that quantifiers do not appear to the immediate right of arrows. Therefore we define the sets

$$\begin{aligned} \mathbf{S} \quad &= \quad \mathbf{S}' \cup \{(\forall t \sigma) \mid \sigma \in \mathbf{S}\}, \\ \mathbf{S}' \quad &= \quad \mathbf{T}_0 \cup \{(\sigma \to \tau) \mid \sigma \in \mathbf{S}, \tau \in \mathbf{S}')\}. \end{aligned}$$

$$(\text{VAR}) \quad A \cup \{x : \sigma\} \vdash x : \tau \quad (\text{where } \sigma \succ \tau)$$

$$(\text{ABS}) \quad \frac{A_x \cup \{x : \tau_1\} \vdash M : \tau_2}{A \vdash (\lambda x M) : \tau_1 \to \tau_2}$$

$$(\text{APP}) \quad \frac{A \vdash M : (\forall \vec{t}\tau_1) \to \tau_2, \quad A \vdash N : \tau_1}{A \vdash (MN) : \tau_2} \quad (\text{each } t_i \notin \text{FTV}(A))$$

Figure 4: Typing rules of Λ_2^{s}. Types in type environments are in $\mathbf{S}(1)$, and derived types are in $\mathbf{S}'(2)$.

We write $\mathbf{S}(n)$ for $\mathbf{S} \cap \mathbf{R}(n)$ and $\mathbf{S}'(n)$ for $\mathbf{S}' \cap \mathbf{R}(n)$. Note that the $\mathbf{S}(1)$ types are exactly the ML type schemes.

Definition 5 Suppose $\sigma = \forall t_1 \cdots t_n.\tau \in \mathbf{S}(1)$, and $\tau, \tau' \in \mathbf{T}_0$. We say τ' *is an instance of* σ, written $\sigma \succ \tau'$, if and only if for some $\rho_1, \ldots, \rho_n \in \mathbf{T}_0$, we have $\tau' = \{t_1 := \rho_1, \ldots, t_n := \rho_n\}\tau$. We write $\sigma \succ (\forall s_1 \cdots s_m \tau')$ if and only if $s_1, \ldots, s_m$ are not free in σ and $\sigma \succ \tau'$.

Note that the sense of '$\succ$' is opposite to that of our other subtyping relations; for example, both "$\sigma \leq_2 \tau$" and "$\sigma \succ \tau$" may be read, "σ is more general than τ." We make an exception in the case of '$\succ$' to be consistent with its use in ML [24].

We now define Λ_2^{s}, our version of the rank 2 fragment of System F. The superscript 's' in Λ_2^{s} indicates that the system is syntax-directed. See Kfoury and Tiuryn [12] for a definition of Λ_2, the non-syntax-directed version.

The judgments of the system are defined by the rules of Figure 4. We write $\Lambda_2^{\mathrm{s}} \triangleright A \vdash M : \tau$ if $A \vdash M : \tau$ is derivable from these rules, where types in type environments are restricted to $\mathbf{S}(1)$, and derived types are restricted to $\mathbf{S}'(2)$.

Λ_2^{s} is closely related to the system Λ_2^- studied by Kfoury et al. [12, 17]:

Theorem 6

i) *If* $\Lambda_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$, *then* $\Lambda_2^- \triangleright A \vdash M : \sigma$.

ii) *If* $\Lambda_2^- \triangleright A \vdash M : \sigma$, *then* σ *is of the form* $\forall t_1 \cdots t_n \sigma'$, *where* $\sigma' \in \mathbf{S}'(2)$, *and* $\Lambda_2^{\mathrm{s}} \triangleright A \vdash M : \sigma'$.

$$\text{(VAR)} \quad A \cup \{x : \sigma\} \vdash x : \tau \quad (\text{where } \sigma \succ \tau)$$

$$\text{(APP)} \quad \frac{A \vdash M : \tau_1 \to \tau_2, \quad A \vdash N : \tau_1}{A \vdash (MN) : \tau_2}$$

$$\text{(ABS)} \quad \frac{A_x \cup \{x : \tau_1\} \vdash M : \tau_2}{A \vdash (\lambda x M) : \tau_1 \to \tau_2}$$

$$\text{(LET)} \quad \frac{A \vdash M_1 : \tau_1, \quad A_x \cup \{x : \mathrm{Gen}(A, \tau_1)\} \vdash M_2 : \tau_2}{A \vdash (\textbf{let } x = M_1 \textbf{ in } M_2) : \tau_2}$$

Figure 5: Typing rules of ML. Types in type environments are in $\mathbf{S}(1)$, and derived types are in $\mathbf{T}_0$.

This equivalence follows immediately from results of Kfoury and Wells [17]. It implies the following useful result:

Lemma 7 *If* $\Lambda_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$ *and* $\mathrm{Gen}(A, \sigma) \succ \sigma'$, *then* $\Lambda_2^{\mathrm{s}} \triangleright A \vdash M : \sigma'$.

2.4 ML

Many different formulations of the ML type system have been studied; we choose to present a syntax-directed version here, as in Clement et al. [4] or Tofte [32].

The *types of ML* are the types $\mathbf{T}_0$, and the *ML type schemes* are the types $\mathbf{S}(1)$. The *terms of ML* are the terms of the lambda calculus extended with *let-expressions*:

$$M \quad ::= \quad x \mid (M_1 M_2) \mid (\lambda x M) \mid (\textbf{let } x = M_1 \textbf{ in } M_2).$$

The judgments of ML are defined inductively by the rules of Figure 5. We write $\mathrm{ML} \triangleright A \vdash M : \tau$ if $A \vdash M : \tau$ is derivable from these rules, where types in type environments are restricted to $\mathbf{S}(1)$, and derived types are restricted to $\mathbf{T}_0$.

Definition 8 An ML type τ is a *principal type for M in A* if and only if $\mathrm{ML} \triangleright A \vdash M : \tau$, and for all ML types τ', if $\mathrm{ML} \triangleright A \vdash M : \tau'$, then $\mathrm{Gen}(A, \tau) \succ \tau'$.

Theorem 9 (Principal types for ML) *If M is typable by A, then there exists a principal type for M in A.*

Lemma 10 *If* $\mathrm{ML} \triangleright A \vdash M : \tau$, *and* $\mathrm{Gen}(A, \tau) \succ \tau'$, *then* $\mathrm{ML} \triangleright A \vdash M : \tau'$.

2.5 Relationship of Λ_2^{s} and $\mathbf{I}_2^{\mathrm{s}}$

We now show that a term is typable in Λ_2^{s} if and only if it is typable in $\mathbf{I}_2^{\mathrm{s}}$. The left to right implication is developed first.

Definition 11

i) We define a relation $\preceq_1$ between $\mathbf{S}(1)$ and $\mathbf{T}_1$ as follows. Suppose $\tau \in \mathbf{S}(1)$ and $\sigma_1, \ldots, \sigma_n \in \mathbf{T}_0$ $(n \geq 1)$. Then $\tau \preceq_1 (\bigwedge_{i \in I} \sigma_i)$ if and only if $\tau \succ \sigma_i$ for all $i \in I$.

ii) We define the relation $\preceq_2$ between $\mathbf{S}'(2)$ and $\mathbf{T}_2$ inductively:

   a) For any type variable t, $t \preceq_2 t$.

   b) If $\tau \preceq_1 \tau'$ and $\sigma \preceq_2 \sigma'$, then $(\tau \rightarrow \sigma) \preceq_2 (\tau' \rightarrow \sigma')$.

Note that the relation $\preceq_2$ is monotonic in the argument of function types, in contrast to the relation $\leq_2$. We extend the relation $\preceq_1$ to type environments as follows: $A \preceq_1 A'$ if and only if $x \in \mathbf{dom}(A)$ and $A(x) \preceq_1 A'(x)$ whenever $x \in \mathbf{dom}(A')$. Note that $A \preceq_1 (A' + A'')$ if $A \preceq_1 A'$ and $A \preceq_1 A''$, and $A \preceq_1 A'$ if $A_x \preceq_1 A'$.

Theorem 12 *If* $\Lambda_2^{\mathrm{s}} \triangleright A \vdash M : \tau$, *then* $\mathbf{I}_2^{\mathrm{s}} \triangleright A' \vdash M : \tau'$, *where* $A \preceq_1 A'$ *and* $\tau \preceq_2 \tau'$.

Proof: By induction on derivations.

i) $M = x$ and $\Lambda_2^{\mathrm{s}} \triangleright A \vdash x : \tau$ follows by the Λ_2^{s} rule (VAR). Then we must have $A(x) \succ \tau$.

   Let $A' = \{x : \tau\}$. Clearly $\mathbf{I}_2^{\mathrm{s}} \triangleright A' \vdash M : \tau$, $A \preceq_1 A'$, and $\tau \preceq_2 \tau$.

ii) $M = \lambda x N$, $\tau = \sigma \rightarrow \tau_1$, and $\Lambda_2^{\mathrm{s}} \triangleright A \vdash \lambda x N : \sigma \rightarrow \tau_1$ follows by the Λ_2^{s} rule (ABS).

   Then we must have

$$\Lambda_2^{\mathrm{s}} \triangleright A_x \cup \{x : \sigma\} \vdash N : \tau_1.$$

   By induction, we have

$$\mathbf{I}_2^{\mathrm{s}} \triangleright A' \cup \{x : \sigma'\} \vdash N : \tau_1',$$

where $A_x \preceq_1 A'$, $\sigma \preceq_1 \sigma'$, and $\tau_1 \preceq_2 \tau_1'$. So by the $\mathbf{I}_2^{\mathrm{s}}$ rule (ABS), we have

$$\mathbf{I}_2^{\mathrm{s}} \rhd A' \vdash N : \sigma' \to \tau_1',$$

where $A \preceq_1 A'$, and $(\sigma \to \tau_1) \preceq_2 (\sigma' \to \tau_1')$, as desired.

iii) $M = M_1 M_2$ and $\Lambda_2^{\mathrm{s}} \rhd A \vdash M_1 M_2 : \tau$ follows by the Λ_2^{s} rule (APP). Then we must have, for some $\tau_0 \in \mathbf{T}_0$,

$$\begin{array}{l} \Lambda_2^{\mathrm{s}} \rhd A \vdash M_1 : (\forall \vec{t}.\tau_0) \to \tau, \\ \Lambda_2^{\mathrm{s}} \rhd A \vdash M_2 : \tau_0, \end{array}$$

where the type variables $\vec{t}$ do not appear in $\mathrm{FTV}(A)$. Then by induction we have

$$\mathbf{I}_2^{\mathrm{s}} \rhd A_0' \vdash M_1 : (\bigwedge_{i \in I} \tau_i) \to \tau',$$

where $A \preceq_1 A_0'$, $\tau \preceq_2 \tau'$, and $(\forall \vec{t}.\tau_0) \preceq_1 (\bigwedge_{i \in I} \tau_i)$.

Then each τ_i is an instance of $(\forall \vec{t}.\tau_0)$, and therefore by Lemma 7, $\Lambda_2^{\mathrm{s}} \rhd A \vdash M_2 : \tau_i$ for all $i \in I$.

By induction we have for all $i \in I$, $\mathbf{I}_2^{\mathrm{s}} \rhd A_i' \vdash M_2 : \tau_i$, where $A \preceq_1 A_i'$. So if $A' = A_0' + \sum_{i \in I} A_i'$, then $A \preceq_1 A'$, and by weakening,

$$\begin{array}{ll} \mathbf{I}_2^{\mathrm{s}} \rhd A' \vdash M_1 : (\bigwedge_{i \in I} \tau_i) \to \tau', & \\ \mathbf{I}_2^{\mathrm{s}} \rhd A' \vdash M_2 : \tau_i & (\forall i \in I). \end{array}$$

Then by the $\mathbf{I}_2^{\mathrm{s}}$ rule (APP) we have

$$\mathbf{I}_2^{\mathrm{s}} \rhd A' \vdash M_1 M_2 : \tau',$$

as desired.

□

We now show the other direction of the equivalence: any term typable in $\mathbf{I}_2^{\mathrm{s}}$ is typable in Λ_2^{s}.

Convention 13 In the remainder of this section we do not consider terms to be identical modulo α-conversion, and we will assume the following convention regarding the names of bound and free variables:

i) No variable is bound more than once.

ii) The bound and free variables are disjoint.

This convention is necessary to make the following function well-defined:

Definition 14 Let ϵ denote the empty sequence. The function, **act**, that maps terms to sequences of variables, is defined inductively by the following rules.[3]

i) $\mathbf{act}(x) = \epsilon$.

ii) If $\mathbf{act}(M) = x_1, \ldots, x_n$ then $\mathbf{act}(\lambda y M) = y, x_1, \ldots, x_n$.

iii) If $\mathbf{act}(M) = y, x_1, \ldots, x_n$ $(n \geq 0)$ then $\mathbf{act}(MN) = x_1, \ldots, x_n$.

iv) If $\mathbf{act}(M) = \epsilon$ then $\mathbf{act}(MN) = \epsilon$.

Definition 15

i) γ is the rule
$$(\lambda x(\lambda y M))N \to \lambda y((\lambda x M)N).$$

ii) $\to_\gamma$ is the compatible closure of γ.

iii) A γ-redex is any term matching the left-hand side of the rule γ. We say M is a γ-normal form, or γ-nf, if no subterm of M is a γ-redex.

Note that by our convention on the distinct naming of variables, there is no capture of variables in the γ rule. We use the name "γ" in accordance with Kfoury and Wells [18]. See Barendregt [2] for a definition of "compatible."

Lemma 16

i) $\to_\gamma$ *is strongly normalizing.*

ii) $\to_\gamma$ *satisfies the diamond property.*

iii) γ*-nf's are unique.*

Proof:

i) The proof is similar to the proof of Lemma 5.5 from Kfoury and Wells [17]:

[3] Our definition is identical to the definition of [12], but differs from [11].

Let $\mathbf{appl}(M)$ be the set of subterms of M that are applications, and let

$$\delta(M) = \sum_{(M_1 M_2) \in \mathbf{appl}(M)} \max(0, |\mathbf{act}(M_1)| - 1).$$

If $M \to_\gamma N$, then $\delta(M) = \delta(N) + 1$. Since for any M we have $\delta(M) \geq 0$, we can conclude that $\to_\gamma$ is strongly normalizing. In fact, $\delta(M) > 0$ iff M contains a γ-redex, and M normalizes in exactly $\delta(M)$ steps.

If $|M|$ is the size (number of subterms) of M, then clearly $|\mathbf{appl}(M)| \leq |M|$ and $|\mathbf{act}(M)| \leq |M|$. Thus $\delta(M) \leq |M|^2$. Therefore normalization of a term M takes $O(|M|^2)$ steps.

ii) This is a simple case analysis.

iii) This follows from (ii).

□

Lemma 16 justifies the following definition:

Definition 17 We write γ-nf(M) for the γ-nf of M.

Lemma 18 *For $\mathcal{D} \in \{\mathbf{I}_2^{\mathrm{s}}, \Lambda_2^{\mathrm{s}}\}$, the following hold:*

i) $\mathcal{D} \triangleright A \vdash (\lambda x(\lambda y M))N : \sigma$ *iff* $\mathcal{D} \triangleright A \vdash \lambda y((\lambda x M)N) : \sigma$.

ii) *If $M \to_\gamma N$, then* $\mathcal{D} \triangleright A \vdash M : \sigma$ *iff* $\mathcal{D} \triangleright A \vdash N : \sigma$.

iii) $\mathcal{D} \triangleright A \vdash M : \sigma$ *iff* $\mathcal{D} \triangleright A \vdash \gamma\text{-nf}(M) : \sigma$.

Proof:

i) Simple case analysis.

ii) Use (i) and induction on the definition of compatible.

iii) Use (ii) and induction on the length of rewriting.

□

Lemma 19 *If $\mathbf{act}(M) = x_1, \ldots, x_n$ and $\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$, then σ is of the form $\sigma_1 \to \cdots \to \sigma_n \to \tau$, where $\tau \in \mathbf{T}_0$.*

Proof: By induction on the structure of M.

i) If $M = x$, then $n = 0$ by the definition of **act**, and $\sigma \in \mathbf{T}_0$ by rule (VAR).

ii) If $M = \lambda x_1 N$, then $\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$ follows by rule (ABS), and therefore σ is of the form $\sigma_1 \to \sigma'$, where $\sigma_1 \in \mathbf{T}_1$.

Also we must have $\mathbf{act}(N) = x_2, \ldots, x_n$ $(n \geq 1)$ and $\mathbf{I}_2^{\mathrm{s}} \triangleright A \cup \{x_1 : \sigma_1\} \vdash N : \sigma'$. By induction σ' must be of the form $\sigma_2 \to \cdots \to \sigma_n \to \tau$, where $\sigma_2, \ldots, \sigma_n \in \mathbf{T}_1$ and $\tau \in \mathbf{T}_0$.

iii) If $M = M_1 M_2$, then $\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$ follows by rule (APP), and therefore we have $\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash M_1 : \sigma' \to \sigma$, where $\sigma' \in \mathbf{T}_1$.

We consider two cases. If $\mathbf{act}(M_1) = y, x_1, \ldots, x_n$ for some variable y, then by induction, σ is of the form $\sigma_1 \to \cdots \to \sigma_n \to \tau$, where $\sigma_1, \ldots, \sigma_n \in \mathbf{T}_1$ and $\tau \in \mathbf{T}_0$.

Otherwise $\mathbf{act}(M_1) = \epsilon$, and therefore $\mathbf{act}(M) = \epsilon$, so we only need prove $\sigma \in \mathbf{T}_0$. And by induction, we have $(\sigma' \to \sigma) \in \mathbf{T}_0$, so $\sigma \in \mathbf{T}_0$.

□

Note 20 A similar lemma holds for Λ_2^{s}, c.f. Kfoury et al. [12], Lemma 15.

Lemma 21 *Suppose M is a γ-nf. Then*

$$\mathbf{act}(M) \neq \epsilon \quad \textit{iff} \quad M = \lambda y N \textit{ for some } y, N.$$

Proof: By induction on the structure of M. The cases $M = x$ and $M = \lambda y N$ are trivial, so assume $M = M_1 M_2$. We must show $\mathbf{act}(M) = \epsilon$.

By way of contradiction, assume that $\mathbf{act}(M) = x_1, \ldots, x_n$ $(n \geq 1)$. By the definition of **act**, we must have $\mathbf{act}(M_1) = y, x_1, \ldots, x_n$ for some y. Then $\mathbf{act}(M_1) \neq \epsilon$, so by induction we have $M_1 = \lambda y M_1'$, and $\mathbf{act}(M_1') = x_1, \ldots, x_n$. Since $n \geq 1$, $\mathbf{act}(M_1') \neq \epsilon$, and by induction $M_1' = \lambda x_1 M_1''$. But then M is a γ-redex, contradiction. □

Definition 22 We define a mapping, **ml**, from terms to ML terms:

i) $\mathbf{ml}(x) = x$.

ii) $\mathbf{ml}(\lambda x M) = (\lambda x \; \mathbf{ml}(M))$.

iii) $\mathbf{ml}(M_1 M_2) = \begin{cases} (\mathbf{let}\ x = \mathbf{ml}(M_2)\ \mathbf{in}\ \mathbf{ml}(N)) & \text{if } M_1 = \lambda x N, \\ (\mathbf{ml}(M_1)\ \mathbf{ml}(M_2)) & \text{otherwise.} \end{cases}$

Definition 23

i) A *generalization* of a set $\mathbf{T}$ of simple types is a type $\sigma \in \mathbf{S}(1)$ such that $\sigma \succ \tau$ for every $\tau \in \mathbf{T}$. A generalization σ of $\mathbf{T}$ is the *least common generalization* of $\mathbf{T}$ if $\sigma' \succ \sigma$ for any other generalization σ' of $\mathbf{T}$.

ii) If $(\bigwedge_{i \in I} \tau_i) \in \mathbf{T}_1$, we define $\mathbf{lcg}(\bigwedge_{i \in I} \tau_i)$ to be the least common generalization of $\{\tau_i \mid i \in I\}$. If $\sigma_1, \ldots, \sigma_n \in \mathbf{T}_1$ and $\tau \in \mathbf{T}_0$, then

$$\mathbf{lcg}(\sigma_1 \to \cdots \to \sigma_n \to \tau) = \mathbf{lcg}(\sigma_1) \to \cdots \to \mathbf{lcg}(\sigma_n) \to \tau.$$

The function $\mathbf{lcg}$ is extended to type environments in the usual way.

The use of "least" in the name "least common generalization" is consistent with the relation '$\succ$'. Recall that the sense of '$\succ$' is opposite to that of our other subtyping relations, so that "least" for '$\succ$' means "greatest" for the other relations.

The concept of least common generalizations was developed by Plotkin [26] and Reynolds [27]. They showed that any finite nonempty set of simple types has a least common generalization, and they gave an algorithm to compute it.

Lemma 24 *If M is a γ-nf and $\sigma \in \mathbf{T}_0$, then*

i) $\mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M : \sigma$ *implies* $\mathrm{ML} \rhd \mathbf{lcg}(A) \vdash \mathbf{ml}(M) : \sigma$*; and*

ii) $\Lambda_2^{\mathrm{s}} \rhd A \vdash M : \sigma$ *if and only if* $\mathrm{ML} \rhd A \vdash \mathbf{ml}(M) : \sigma$.

Proof:

i) By induction on the structure of M.

a) The case $M = x$ is trivial.

b) If $M = \lambda y N$, then $\mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M : \sigma$ follows by the $\mathbf{I}_2^{\mathrm{s}}$ rule (ABS), so σ must be of the form $\tau \to \sigma'$ where $\tau, \sigma' \in \mathbf{T}_0$, and $\mathbf{I}_2^{\mathrm{s}} \rhd A \cup \{y : \tau\} \vdash N : \sigma'$. Note that N is a γ-nf, so we can apply the induction hypothesis to get

$$\mathrm{ML} \rhd \mathbf{lcg}(A \cup \{y : \tau\}) \vdash \mathbf{ml}(N) : \sigma'.$$

Now $\tau \in \mathbf{T}_0$, so $\mathbf{lcg}(A \cup \{y : \tau\}) = \mathbf{lcg}(A) \cup \{y : \tau\}$. Therefore $\mathrm{ML} \rhd \mathbf{lcg}(A) \cup \{y : \tau\} \vdash \mathbf{ml}(N) : \sigma'$, so by the ML rule (ABS), $\mathrm{ML} \rhd \mathbf{lcg}(A) \vdash \mathbf{ml}(\lambda y N) : \tau \to \sigma'$, as desired.

c) If $M = (\lambda y M_1)M_2$, then our judgment must follow by the $\mathbf{I}_2^{\mathrm{s}}$ rules (ABS) and (APP). Thus we have

$$\begin{array}{rl} & \mathbf{I}_2^{\mathrm{s}} \rhd A \cup \{y : (\bigwedge_{i\in I} \sigma_i)\} \vdash M_1 : \sigma, \\ (\forall i \in I) & \mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M_2 : \sigma_i. \end{array}$$

Let $\forall \vec{t}\tau = \mathbf{lcg}(\bigwedge_{i\in I} \sigma_i)$, where $\tau \in \mathbf{T}_0$, and no t_i appears in A. By induction, we have

$$\begin{array}{rl} & \mathrm{ML} \rhd \mathbf{lcg}(A) \cup \{y : \forall \vec{t}\tau\} \vdash \mathbf{ml}(M_1) : \sigma, \\ (\forall i \in I) & \mathrm{ML} \rhd \mathbf{lcg}(A) \vdash \mathbf{ml}(M_2) : \sigma_i. \end{array}$$

By the principal type property of ML, we have

$$\mathrm{ML} \rhd \mathbf{lcg}(A) \vdash \mathbf{ml}(M_2) : \tau.$$

Then since $\mathbf{ml}(M) = (\mathbf{let}\ y = \mathbf{ml}(M_2)\ \mathbf{in}\ \mathbf{ml}(M_1))$, we have

$$\mathrm{ML} \rhd \mathbf{lcg}(A) \vdash \mathbf{ml}(M) : \sigma$$

by the ML rule (LET).

d) If $M = M_1 M_2$, where M_1 is not an abstraction, then by the $\mathbf{I}_2^{\mathrm{s}}$ rule (APP), we have for some σ',

$$\begin{array}{l} \mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M_1 : \sigma' \to \sigma, \\ \mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M_2 : \sigma'. \end{array}$$

M_1 is a γ-nf and is not an abstraction, so by Lemma 21, we have $\mathbf{act}(M_1) = \epsilon$. Then by Lemma 19, $\sigma' \to \sigma \in \mathbf{T}_0$, and therefore $\sigma' \in \mathbf{T}_0$. M_2 is also a γ-nf, so we may apply the induction hypothesis to both judgments above, to get

$$\begin{array}{l} \mathrm{ML} \rhd \mathbf{lcg}(A) \vdash \mathbf{ml}(M_1) : \sigma' \to \sigma, \\ \mathrm{ML} \rhd \mathbf{lcg}(A) \vdash \mathbf{ml}(M_2) : \sigma'. \end{array}$$

Then by the ML rule (APP), we have

$$\mathrm{ML} \rhd \mathbf{lcg}(A) \vdash \mathbf{ml}(M_1 M_2) : \sigma,$$

as desired.

ii) Similar, but easier.

□

Note 25 The converse of Lemma 24(i) does not hold. For instance, if $\sigma = t_3$ and $A = \{x : t_1 \wedge t_2\}$, then $\mathbf{lcg}(A) = \{x : \forall t.t\}$, $\mathbf{ml}(xx) = xx$, and $\mathrm{ML} \triangleright \{x : \forall t.t\} \vdash xx : t_3$, but the judgment $\{x : t_1 \wedge t_2\} \vdash xx : t_3$ cannot be derived in $\mathbf{I}_2^{\mathrm{s}}$.

Theorem 26 *If* $\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$, *then* $\Lambda_2^{\mathrm{s}} \triangleright \mathbf{lcg}(A) \vdash M : \mathbf{lcg}(\sigma)$.

Proof: Suppose $\mathbf{act}(M) = x_1, \ldots, x_n$. Then by Lemma 19, σ is of the form $\sigma_1 \to \cdots \to \sigma_n \to \tau$, where $\tau \in \mathbf{T}_0$, and by Lemma 21, the γ-nf of M is of the form $\lambda x_1 \cdots \lambda x_n N$, where N is a γ-nf. By Lemma 18(iii),

$$\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash \lambda x_1 \cdots \lambda x_n N : \sigma.$$

This judgment must follow by n uses of the $\mathbf{I}_2^{\mathrm{s}}$ rule (ABS), so we have

$$\mathbf{I}_2^{\mathrm{s}} \triangleright A \cup \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\} \vdash N : \tau.$$

Then by Lemma 24, we have

$$\Lambda_2^{\mathrm{s}} \triangleright \mathbf{lcg}(A \cup \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}) \vdash N : \tau.$$

By n uses of the Λ_2^{s} rule (ABS), we have

$$\Lambda_2^{\mathrm{s}} \triangleright \mathbf{lcg}(A) \vdash \lambda x_1 \cdots \lambda x_n N : \mathbf{lcg}(\sigma),$$

and by Lemma 18(iii), we have

$$\Lambda_2^{\mathrm{s}} \triangleright \mathbf{lcg}(A) \vdash M : \mathbf{lcg}(\sigma).$$

□

Theorem 27 *If M is a term of the pure lambda calculus, then M is typable in $\mathbf{I}_2^{\mathrm{s}}$ if and only if M is typable in Λ_2^{s}.*

Therefore, typability in $\mathbf{I}_2^{\mathrm{s}}$ is DEXPTIME-complete.

Proof: The equivalence of $\mathbf{I}_2^{\mathrm{s}}$ and Λ_2^{s} typability follows from Theorems 12 and 26.

Kfoury and Tiuryn [12] show that Λ_2^{s} typability is polynomial time equivalent to ML typability. ML typability was shown to be DEXPTIME-complete independently by Kfoury et al. [15] and by Mairson [22]. □

The equivalence of Theorem 27 has been shown independently by Yokouchi [35].

3 Type inference for $\mathbf{I}_2^{\mathrm{s}}$

We present the type inference algorithm for $\mathbf{I}_2^{\mathrm{s}}$ and a proof that it infers principal pairs. The algorithm is not new: it was described briefly in Leivant's original paper [21], and was defined rigorously by van Bakel in his dissertation [33]. We include it here because the algorithm provides a way to compare a variety of type systems based on rank 2 intersection types, and because we will extend the algorithm in a later section.

The algorithm takes as input a term M, and produces a pair $\langle A, \sigma\rangle$ such that $\mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M : \sigma$. Moreover, the pair $\langle A, \sigma\rangle$ is *principal* in the sense that any other acceptable pair of M can be obtained from $\langle A, \sigma\rangle$ by some well-understood operations.

Definition 28

i) We write $A \leq_1 A'$ if $x \in \mathbf{dom}(A)$ and $A(x) \leq_1 A'(x)$ for all $x \in \mathbf{dom}(A')$.

ii) The ordering $\leq$ on ($\mathbf{T}_1$ type environment, $\mathbf{T}_2$ type) pairs is defined as follows:

$$\langle A, \sigma\rangle \leq \langle A', \sigma'\rangle \text{ if and only if } A' \leq_1 A \text{ and } \sigma \leq_2 \sigma'.$$

iii) A pair $\langle A, \sigma\rangle$ is a *principal pair for M* if $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{I}_2^{\mathrm{s}}}(M)$, and for any other pair $\langle A', \sigma'\rangle \in \mathrm{AP}_{\mathbf{I}_2^{\mathrm{s}}}(M)$, there is a substitution S such that $S\langle A, \sigma\rangle \leq \langle A', \sigma'\rangle$.

Note that $\leq_1$ and $\leq$ are transitive, and $A + A' \leq_1 A$ for all $\mathbf{T}_1$ type environments A, A'.

3.1 Subtype satisfaction

In this section we give a decision procedure for one of our subtyping relations, and show how to solve a more general problem, *subtype satisfaction*, that we use in our type inference algorithm.

Up until now, we have relied on some syntactic conventions to simplify our presentation, namely, that '$\wedge$' is an associative, commutative, and idempotent operator. Part of the problem we are addressing here is how to decide whether two types are equivalent under these assumptions. Therefore, in this section, we do not rely on the syntactic conventions in any way.

Subtype satisfaction is a generalization of the well-known problem of *unification*, and the techniques we use here are based on those used to solve unification. For more details, consult a survey on unification [19, 20, 30, 10, 6, 31, 1]. One difference between unification and our satisfaction problems is that we work with types that go beyond simple types, but our substitutions involve only simple types. This is not the typical case with unification, and it makes our problem easier to solve.

If S_1, S_2 are substitutions and V is a set of type variables, we say *S_1 and S_2 are equivalent on V*, written $S_1 =_V S_2$, if $S_1 t = S_2 t$ for every $t \in V$. We say *S_1 is more general than S_2 on V*, written $S_1 \leq_V S_2$, if there is a substitution S_3 such that $S_2 =_V S_3 S_1$. The relation $\leq_V$ is a partial order modulo $=_V$. We omit V when $V = \mathbf{Tv}$. A substitution S is *idempotent* if $S = SS$, or, equivalently, if $\mathbf{dom}(S) \cap \mathbf{rng}(S) = \varnothing$.

We define the relation $\leq_{2,1}$ between $\mathbf{T}_2$ and $\mathbf{T}_1$ to be the least relation closed under the rule:

- If $\sigma \leq_2 \tau_i$ for all $i \in I$, then $\sigma \leq_{2,1} (\bigwedge_{i \in I} \tau_i)$.

A *$\leq_{2,1}$-satisfaction problem* is a pair $\exists \vec{s}.P$, where P is a finite set whose every element is either: 1) an equality between simple types; or 2) an inequality between a $\mathbf{T}_2$ type and a $\mathbf{T}_1$ type. When $\vec{s}$ is empty $\exists \vec{s}$ may be omitted. We use π to range over $\leq_{2,1}$-satisfaction problems.

A substitution S is a *solution* to $\exists \vec{s}.P$ if there is a substitution S' such that $S(t) = S'(t)$ for all $t \notin \vec{s}$, $S'\sigma \leq_{2,1} S'\tau$ for all inequalities $(\sigma \leq \tau) \in P$, and $S'\sigma = S'\tau$ for all equalities $(\sigma = \tau) \in P$. The (possibly empty) set of solutions to a problem π is written $\mathbf{Solutions}(\pi)$. Two problems π_1 and π_2 are *equivalent* if $\mathbf{Solutions}(\pi_1) = \mathbf{Solutions}(\pi_2)$.

Definition 29

i) A substitution U is a *most general solution to* π if it satisfies the following conditions.

   a) $U \in \mathbf{Solutions}(\pi)$.
   b) If $S \in \mathbf{Solutions}(\pi)$ then $U \leq_{\mathrm{FTV}(\pi)} S$.
   c) U is idempotent.
   d) $\mathbf{dom}(U) \subseteq \mathrm{FTV}(\pi)$.

ii) We write $\mathbf{MGS}(\pi)$ for the (possibly empty) set of most general solutions to a $\leq_{2,1}$-satisfaction problem π.

We require the last two conditions on most general solutions for technical convenience only. We could relax the definition by eliminating those conditions; but any π has a solution under the relaxed definition if and only if it has a solution under our definition.

Sometimes it is useful to ensure that a most general solution does not interfere with a set of "protected" variables. For any set W of type variables, we say U is a *most general solution to* π *away from* W if $U \in \mathbf{MGS}(\pi)$ and $W \cap \mathbf{rng}(U) = \emptyset$, and we write $\mathbf{MGS}(\pi)[W]$ for the (possibly empty) set of most general solutions to π away from W.

Lemma 30 *If* $U \in \mathbf{MGS}(\pi)[W]$ *and* $S \in \mathbf{Solutions}(\pi)$*, then* $U \leq_{W \cup \mathrm{FTV}(\pi)} S$.

Proof: Since $U \leq_{\mathrm{FTV}(\pi)} S$, there is some R such that $RU =_{\mathrm{FTV}(\pi)} S$. Define

$$R'(t) = \begin{cases} R(t) & \text{if } t \in \mathbf{rng}(U), \\ S(t) & \text{otherwise.} \end{cases}$$

If $t \in \mathrm{FTV}(\pi)$, then $R'(U(t)) = R(U(t)) = S(t)$. And if $t \in W - \mathrm{FTV}(\pi)$, then $t \notin (\mathbf{dom}(\pi) \cup \mathbf{rng}(\pi))$, so $R'(U(t)) = R'(t) = S(t)$. □

A *unification problem* is a subtype satisfaction problem involving only equalities. Algorithms for solving unification problems are well known; in particular, we have the following theorem.

Theorem 31 *Let* π *be a unification problem and* W *be a finite set of type variables.*

i) $\mathbf{Solutions}(\pi) = \emptyset$ *iff* $\mathbf{MGS}(\pi) = \emptyset$ *iff* $\mathbf{MGS}(\pi)[W] = \emptyset$.

ii) *There is an algorithm that decides whether* π *has a solution, and, if so, returns an element of* $\mathbf{MGS}(\pi)[W]$.

Proof: See for example Snyder [31], Lemma 3.3.11. □

Theorem 32 *Every* $\leq_{2,1}$*-satisfaction problem is equivalent to a unification problem, and moreover, there is an algorithm that transforms every* $\leq_{2,1}$*-satisfaction problem into an equivalent unification problem.*

Corollary 33 *Let* π *be a* $\leq_{2,1}$*-satisfaction problem and* W *be a finite set of type variables.*

i) $\mathbf{Solutions}(\pi) = \emptyset$ *iff* $\mathbf{MGS}(\pi) = \emptyset$ *iff* $\mathbf{MGS}(\pi)[W] = \emptyset$.

ii) *There is an algorithm that decides whether* π *has a solution, and, if so, returns an element of* $\mathbf{MGS}(\pi)[W]$.

$$\begin{array}{lll}
(\sigma_1 \rightarrow \sigma_2) \leq t & \Rightarrow & \exists t_1, t_2 . \{t_1 \leq \sigma_1, \sigma_2 \leq t_2, t = t_1 \rightarrow t_2\} \\
 & & \text{if } t_1, t_2 \text{ are fresh} \\
(\sigma_1 \rightarrow \sigma_2) \leq (\tau_1 \rightarrow \tau_2) & \Rightarrow & \{\tau_1 \leq \sigma_1, \sigma_2 \leq \tau_2\} \\
\sigma \leq (\tau_1 \wedge \tau_2) & \Rightarrow & \{\sigma \leq \tau_1, \sigma \leq \tau_2\} \\
t \leq \tau & \Rightarrow & \{t = \tau\} \\
 & & \text{if } \tau \text{ is a simple type}
\end{array}$$

Figure 6: Transformational rules for $\leq_{2,1}$-satisfaction problems

We will prove Theorem 32 by giving an algorithm that transforms any $\leq_{2,1}$-satisfaction problem into an equivalent unification problem. Corollary 33 follows by combining the transformation with any unification algorithm.

Our transformation is defined by rules of the form

$$\sigma \leq \tau \quad \Rightarrow \quad \exists \vec{t}. P.$$

The rules may need to introduce fresh type variables, that is, type variables that do not appear on the left-hand side. These variables will appear in the variables $\vec{t}$ of the right-hand side (but they are not the only source of variables in $\vec{t}$).

The rules are used to define a rewrite relation on problems:

$$\frac{\sigma \leq \tau \quad \Rightarrow \quad \exists \vec{t}. P}{\exists \vec{s}. P' \uplus \{\sigma \leq \tau\} \quad \Rightarrow \quad \exists \vec{s} \uplus \vec{t}. P' \cup P}$$

The operator '$\uplus$' is disjoint union; on the right of the consequent, it means that the variables $\vec{t}$ must be fresh (this can always be achieved by renaming).

The rules for transforming a $\leq_{2,1}$-satisfaction problem into a unification problem are given in Figure 6.

Proof of Theorem 32: We show that the rules of Figure 6 constitute an algorithm for converting any $\leq_{2,1}$-satisfaction problem into an equivalent unification problem.

First, note that every rule transforms a $\leq_{2,1}$-satisfaction problem into another $\leq_{2,1}$-satisfaction problem (equalities are between simple types, inequalities are between $\mathbf{T}_2$ and $\mathbf{T}_1$ types).

Second, note that each rule preserves the set of solutions, so that each application of a rule transforms a problem into an equivalent problem.

Third, note that repeated application of these rules must halt: every rule reduces the number of type constructors ('$\rightarrow$' or '$\wedge$') in inequalities or reduces the number of inequalities.

Finally, note that a normal form contains no inequalities, and is therefore a unification problem. $\square$

Theorem 34 *The subtyping relation $\leq_{2,1}$ is decidable.*

Proof: To see whether $\sigma \leq_{2,1} \tau$, compute $U \in \mathbf{MGS}(\{\sigma \leq \tau\})$ and check to see whether U is the identity substitution. $\square$

Decision procedures for the other subtyping relations can be obtained in a similar way.

Because we so often want to ensure that $U \in \mathbf{MGS}(\pi)$ is chosen "away" from a set of type variables, we adopt the following convention.

Convention 35 Whenever $U \in \mathbf{MGS}(\pi)$ occurs in any mathematical context, we assume that U is chosen so that it does not interfere with "current" type variables, that is, $U \in \mathbf{MGS}(\pi)[W]$ where $W \cup \mathrm{FTV}(\pi)$ is the set of type variables present in the context.

3.2 Type inference

Definition 36 For any term M, we define the set $\mathrm{PP}_{\mathbf{I}_2^s}(M)$ of pairs by induction:

i) If $M = x$, then for any type variable t, $\langle \{x : t\}, t\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(x)$.

ii) If $M = \lambda x N$, and $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(N)$, then:

   a) If $x \notin \mathbf{dom}(A)$, and t is a type variable not appearing in $\langle A, \sigma\rangle$, then $\langle A, t \rightarrow \sigma\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(\lambda x N)$.

   b) If $x \in \mathbf{dom}(A)$, then $\langle A_x, A(x) \rightarrow \sigma\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(\lambda x N)$.

iii) If $M = M_1 M_2$, then:

   a) If $\langle A_1, t\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1)$ and $\langle A_2, \sigma_2\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_2)$ are disjoint, and $U \in \mathbf{MGS}(\{t = t_1 \rightarrow t_2, \sigma_2 \leq t_1\})$ where t_1, t_2 are fresh, then

$$U\langle A_1 + A_2, t_2\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1 M_2).$$

b) If $\langle A_1, (\bigwedge_{i\in I} \sigma_i) \to \sigma_1\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1)$, and $\langle A_i, \tau_i\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_2)$ for all $i \in I$, where all pairs are chosen disjoint, and $U \in \mathbf{MGS}(\{\tau_i \leq \sigma_i \mid i \in I\})$, then

$$U\langle A_1 + \sum_{i\in I} A_i, \sigma_1\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1 M_2).$$

The following lemma establishes that the elements of $\mathrm{PP}_{\mathbf{I}_2^s}(M)$ are just trivial variants of each other. Therefore, the requirement of disjointness used in the definition of $\mathrm{PP}_{\mathbf{I}_2^s}$ is easily satisfied, and Definition 36 can be adapted to a type inference algorithm.

Lemma 37

i) *If $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M)$, then $x \in \mathbf{dom}(A)$ if and only if x is free in M.*

ii) *Suppose $\langle A_1, \sigma_1\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M)$. Then $\langle A_2, \sigma_2\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M)$ if and only if there is a bijection R of type variables such that $R\langle A_1, \sigma_1\rangle = \langle A_2, \sigma_2\rangle$.*

Proof: An easy induction on Definition 36. □

Theorem 38 *There is an algorithm that decides, for any M, whether the set $\mathrm{PP}_{\mathbf{I}_2^s}(M)$ is empty; and furthermore, if $\mathrm{PP}_{\mathbf{I}_2^s}(M)$ is not empty, it produces a member of $\mathrm{PP}_{\mathbf{I}_2^s}(M)$.*

Proof: Just follow the rules of Definition 36, generating "fresh" type variables as necessary, and use the algorithm of Corollary 33 to compute **MGS**. □

Example 39 We show how the algorithm finds the principal pair of $(\lambda x.xx)$.

i) $\mathrm{PP}_{\mathbf{I}_2^s}(x)$ produces a pair $\langle\{x : t_1\}, t_1\rangle$.

ii) $\mathrm{PP}_{\mathbf{I}_2^s}(x)$ (again) produces a pair $\langle\{x : t_2\}, t_2\rangle$.

iii) To calculate $\mathrm{PP}_{\mathbf{I}_2^s}(xx)$, we find a most general solution to

$$\{t_2 \leq t_3, t_1 = t_3 \to t_4\},$$

such as $\{t_2 := t_3, t_1 := t_3 \to t_4\}$. Then $\langle\{x : t_3 \wedge (t_3 \to t_4)\}, t_4\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(xx)$.

iv) Finally, $\mathrm{PP}_{\mathbf{I}_2^s}(\lambda x.xx)$ produces $\langle\varnothing, t_3 \wedge (t_3 \to t_4) \to t_4\rangle$.

We now establish the soundness of $\mathrm{PP}_{\mathbf{I}_2^s}$.

Theorem 40 *If* $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M)$, *then* $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{I}_2^s}(M)$.

Proof: By induction on the definition of $\mathrm{PP}_{\mathbf{I}_2^s}(M)$.

i) If $M = x$, then $\langle A, \sigma\rangle = \langle\{x : t\}, t\rangle$, and we have $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{I}_2^s}(x)$ by rule (VAR).

ii) If $M = \lambda xN$, then by Lemma 37(i) we have the following two cases:

   a) x is not free in N, and $\sigma = t \to \sigma'$, where $\langle A, \sigma'\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(N)$.
   By induction and weakening, $\langle A \cup \{x : t\}, \sigma'\rangle \in \mathrm{AP}_{\mathbf{I}_2^s}(N)$ (note that $A \cup \{x : t\}$ is well-formed by Lemma 37(i)).
   So by rule (ABS), $\langle A, t \to \sigma'\rangle = \langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{I}_2^s}(\lambda xN)$.

   b) x is free in N and $\langle A, \sigma\rangle = \langle A'_x, A'(x) \to \sigma'\rangle$, where $\langle A', \sigma'\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(N)$.
   By induction $\langle A', \sigma'\rangle \in \mathrm{AP}_{\mathbf{I}_2^s}(N)$, so $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{I}_2^s}(\lambda xN)$ by rule (ABS).

iii) If $M = M_1M_2$, then one of the following cases holds:

   a) $\langle A, \sigma\rangle = U\langle A_1 + A_2, t_2\rangle$, where $\langle A_1, t\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1)$, $\langle A_2, \sigma_2\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_2)$, and $U \in \mathbf{MGS}(\{t = t_1 \to t_2, \sigma_2 \leq t_1\})$.
   Then by induction, weakening, and substitutivity,

$$\begin{aligned} U\langle A_1 + A_2, t\rangle &\in \mathrm{AP}_{\mathbf{I}_2^s}(M_1), \\ U\langle A_2 + A_2, \sigma_2\rangle &\in \mathrm{AP}_{\mathbf{I}_2^s}(M_2). \end{aligned}$$

   Since $U\sigma_2 \leq_2 Ut_1$, by Lemma 2(ii) we have $U\sigma_2 = Ut_1$. And $Ut = (Ut_1) \to (Ut_2)$, so by rule (APP) we have $U\langle A_1 + A_2, t_2\rangle \in \mathrm{AP}_{\mathbf{I}_2^s}(M)$.

   b) $\langle A, \sigma\rangle = U\langle A_1 + \sum_{i\in I} A_i, \sigma_1\rangle$, where $\langle A_i, \tau_i\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_2)$ for all $i \in I$, $\langle A_1, (\bigwedge_{i\in I} \sigma_i) \to \sigma_1\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1)$, and $U \in \mathbf{MGS}(\{\tau_i \leq \sigma_i \mid i \in I\})$.
   Then by induction, weakening, and substitutivity,

$$\begin{aligned} U\langle A_1 + \textstyle\sum_{i\in I} A_i, (\bigwedge_{i\in I} \sigma_i) \to \sigma_1\rangle &\in \mathrm{AP}_{\mathbf{I}_2^s}(M_1), \\ U\langle A_1 + \textstyle\sum_{i\in I} A_i, \tau_i\rangle &\in \mathrm{AP}_{\mathbf{I}_2^s}(M_2) \quad (\forall i \in I). \end{aligned}$$

   By Lemma 2(ii) and the fact that $U\tau_i \leq_2 U\sigma_i$, we have $U\tau_i = U\sigma_i$. Then by (APP), $U\langle A_1 + \sum_{i\in I} A_i, \sigma_1\rangle \in \mathrm{AP}_{\mathbf{I}_2^s}(M)$. □

Theorem 41 (Principal pairs for $\mathbf{I}_2^{\mathrm{s}}$) *If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{I}_2^{\mathrm{s}}}(M)$, then there is a pair $\langle A', \sigma'\rangle \in \mathrm{PP}_{\mathbf{I}_2^{\mathrm{s}}}(M)$ and a substitution S such that $S\langle A', \sigma'\rangle \leq \langle A, \sigma\rangle$.*

Proof: By cases on the structure of M.

i) If $M = x$, then $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{I}_2^{\mathrm{s}}}(M)$ by rule (VAR), and therefore, $A(x) = (\bigwedge_{i\in I} \sigma_i)$ and $\sigma = \sigma_{i_0} \in \mathbf{T}_0$ for some $i_0 \in I$.

   For any t, $\langle\{x : t\}, t\rangle \in \mathrm{PP}_{\mathbf{I}_2^{\mathrm{s}}}(M)$. Then $\{t := \sigma\}$ is a well-formed substitution and

   $$\{t := \sigma\}\langle\{x : t\}, t\rangle = \langle\{x : \sigma\}, \sigma\rangle \leq \langle A, \sigma\rangle.$$

ii) If $M = \lambda x N$, then by the definition of $\mathbf{I}_2^{\mathrm{s}}$, σ must be of the form $\sigma_1 \to \sigma_2$, and $\langle A_x \cup \{x : \sigma_1\}, \sigma_2\rangle \in \mathrm{AP}_{\mathbf{I}_2^{\mathrm{s}}}(N)$. By induction, there is a substitution S and pair $\langle A', \sigma_2'\rangle \in \mathrm{PP}_{\mathbf{I}_2^{\mathrm{s}}}(N)$ such that

   $$S\langle A', \sigma_2'\rangle \leq \langle A_x \cup \{x : \sigma_1\}, \sigma_2\rangle. \tag{1}$$

   We consider two cases.

   a) If $x \notin \mathbf{dom}(A')$, then for any fresh type variable t, $\langle A', t \to \sigma_2'\rangle \in \mathrm{PP}_{\mathbf{I}_2^{\mathrm{s}}}(\lambda x N)$.
      Note that σ_1 is of the form $(\bigwedge_{i\in I} \sigma_i)$, and therefore, we can pick $\sigma_1' \in \mathbf{T}_0$ such that $\sigma_1 \leq_1 \sigma_1'$ (choose any σ_i). Then let $S' = \{t := \sigma_1'\} \cup S$. By (1) and the definition of $\leq$,

      $$S'\langle A', t \to \sigma_2'\rangle = \langle SA', \sigma_1' \to S\sigma_2'\rangle \leq \langle A_x, \sigma_1 \to \sigma_2\rangle.$$

      Since $A \leq_1 A_x$, we have $S'\langle A', t \to \sigma_2'\rangle \leq \langle A, \sigma_1 \to \sigma_2\rangle$, as desired.

   b) If $x \in \mathbf{dom}(A')$, then $\langle A_x', A'(x) \to \sigma_2'\rangle \in \mathrm{PP}_{\mathbf{I}_2^{\mathrm{s}}}(\lambda x N)$. Then by (1) and the definition of $\leq$,

      $$S\langle A_x', A'(x) \to \sigma_2'\rangle \leq \langle A_x, \sigma_1 \to \sigma_2\rangle,$$

      and since $A \leq_1 A_x$, we have $S\langle A_x', A'(x) \to \sigma_2'\rangle \leq \langle A, \sigma_1 \to \sigma_2\rangle$, as desired.

iii) If $M = M_1 M_2$, then by the definition of $\mathbf{I}_2^{\mathrm{s}}$, $\langle A, (\bigwedge_{i\in I} \sigma_i) \to \sigma\rangle \in \mathrm{AP}_{\mathbf{I}_2^{\mathrm{s}}}(M_1)$ and $\langle A, \sigma_i\rangle \in \mathrm{AP}_{\mathbf{I}_2^{\mathrm{s}}}(M_2)$ for all $i \in I$.

   By induction, $\mathrm{PP}_{\mathbf{I}_2^{\mathrm{s}}}(M_1)$ is nonempty, and by Lemma 37(ii), it is sufficient to consider the following cases on the structure of pairs in $\mathrm{PP}_{\mathbf{I}_2^{\mathrm{s}}}(M_1)$.

a) $\langle A_1, t\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1)$. By induction, there is a substitution S_1 such that

$$S_1\langle A_1, t\rangle \quad \leq \quad \langle A, (\bigwedge_{i\in I} \sigma_i) \to \sigma\rangle.$$

By the definition of $\leq_2$, $S_1 t = \sigma_i \to \sigma'$ for some $i \in I$ and $\sigma' \in \mathbf{T}_0$. Then by induction and Lemma 37(ii), there is a disjoint pair $\langle A_2, \tau\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_2)$ and substitution S_2 such that

$$S_2\langle A_2, \tau\rangle \quad \leq \quad \langle A, \sigma_i\rangle.$$

Let $\pi = \{t = t_1 \to t_2, \tau \leq t_1\}$, where t_1, t_2 are fresh. Then $S = S_1 \cup S_2 \cup \{t_1 := \sigma_i, t_2 := \sigma'\}$ is a solution to π.
Pick $U \in \mathbf{MGS}(\pi)$. Then $U\langle A_1 + A_2, t_2\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1 M_2)$.
By Convention 35, there exists an R such that $RU\langle A_1 + A_2, t_2\rangle = S\langle A_1 + A_2, t_2\rangle$. And

$$S\langle A_1 + A_2, t_2\rangle = \langle S_1 A_1 + S_2 A_2, \sigma'\rangle \leq \langle A, \sigma\rangle,$$

as desired.

b) $\langle A_1, (\bigwedge_{j\in J} \sigma'_j) \to \sigma'\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1)$.
By induction there is a substitution S_1 such that

$$S_1\langle A_1, (\bigwedge_{j\in J} \sigma'_j) \to \sigma'\rangle \quad \leq \quad \langle A, (\bigwedge_{i\in I} \sigma_i) \to \sigma\rangle.$$

By the definition of $\leq_2$, $\{S_1\sigma'_j \mid j \in J\} \subseteq \{\sigma_i \mid i \in I\}$, so for all $j \in J$ there is an $i_j \in I$ such that $S_1\sigma'_j = \sigma_{i_j}$.
By induction and Lemma 37(ii), for all $j \in J$ there are disjoint pairs $\langle A_j, \rho_j\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_2)$ and substitutions S_j such that

$$S_j\langle A_j, \rho_j\rangle \quad \leq \quad \langle A, \sigma_{i_j}\rangle.$$

Let $\pi = \{\rho_j \leq \sigma'_j \mid j \in J\}$. Then $S = S_1 \cup (\bigcup_{j\in J} S_j)$ is a solution to π: $S\rho_j = S_j\rho_j \leq_2 \sigma_{i_j} = S_1\sigma'_j = S\sigma'_j$.
Pick $U \in \mathbf{MGS}(\pi)$. Then

$$U\langle A_1 + \sum_{j\in J} A_j, \sigma'\rangle \in \mathrm{PP}_{\mathbf{I}_2^s}(M_1 M_2).$$

By Convention 35, there exists an R such that

$$RU\langle A_1 + \sum_{j\in J} A_j, \sigma'\rangle = S\langle A_1 + \sum_{j\in J} A_j, \sigma'\rangle.$$

And

$$S\langle A_1 + \sum_{j\in J} A_j, \sigma'\rangle = \langle S_1 A_1 + \sum_{j\in J} S_j A_j, S_1\sigma'\rangle \leq \langle A, \sigma\rangle,$$

as desired.

□

4 Other systems of rank 2 intersection types

4.1 A restriction of $\mathbf{I}_2^{\mathrm{s}}$

Van Bakel [33] defined a rank 2 intersection type system that is a slight restriction of our system $\mathbf{I}_2^{\mathrm{s}}$. A version of his rules is presented below.

$$(\text{VAR}) \quad \{x : \tau\} \vdash x : \tau \quad (\text{where } \tau \in \mathbf{T}_0)$$

$$(\text{ABS}) \quad \frac{A_x \cup \{x : \tau_1\} \vdash M : \tau_2}{A_x \vdash (\lambda x M) : \tau_1 \to \tau_2}$$

$$(\text{APP}) \quad \frac{A \vdash M : (\bigwedge_{i\in I} \tau_i) \to \tau, \quad (\forall i \in I)\ A_i \vdash N : \tau_i}{A + \sum_{i\in I} A_i \vdash (MN) : \tau}$$

We write $\mathbf{I}_2^{\mathrm{vb}} \rhd A \vdash M : \sigma$ if the judgment $A \vdash M : \sigma$ follows by these rules, under the following restrictions: environment types are in $\mathbf{T}_1$; derived types are in $\mathbf{T}_2$; and in every judgment $A \vdash M : \tau$, the type environment A contains only assumptions actually used in the derivation of $A \vdash M : \tau$. For example, the rule (VAR) has been intentionally restricted to rule out a judgment such as

$$\{x : \sigma_1 \wedge \sigma_2\} \vdash x : \sigma_1,$$

in which the type σ_2 assumed for x is not used. Similarly, $\{x : \sigma_1, y : \sigma_2\} \vdash x : \sigma_1$ is not derivable because the assumption $y : \sigma_2$ is not used. The exact relation between $\mathbf{I}_2^{\mathrm{vb}}$ and $\mathbf{I}_2^{\mathrm{s}}$ is summed up in the following lemma.

Theorem 42 (Comparison of $\mathbf{I}_2^{\mathrm{vb}}$ and $\mathbf{I}_2^{\mathrm{s}}$)
i) *If* $\mathbf{I}_2^{\mathrm{vb}} \rhd A \vdash M : \sigma$, *then* $\mathbf{I}_2^{\mathrm{s}} \rhd A \vdash M : \sigma$. *The converse does not hold.*

ii) *A term M is typable in* $\mathbf{I}_2^{\mathrm{yb}}$ *if and only if it is typable in* $\mathbf{I}_2^{\mathrm{s}}$.

Proof:

i) Just note that the $\mathbf{I}_2^{\mathrm{yb}}$ rule (VAR) is a special case of the $\mathbf{I}_2^{\mathrm{s}}$ rule (VAR), that the $\mathbf{I}_2^{\mathrm{yb}}$ rule (ABS) is identical to the $\mathbf{I}_2^{\mathrm{s}}$ rule (ABS), and that the $\mathbf{I}_2^{\mathrm{yb}}$ rule (APP) follows from the $\mathbf{I}_2^{\mathrm{s}}$ rule (ABS) and weakening.

   The examples above show that the converse does not hold.

ii) This follows because the definition of principal pair in van Bakel's system is identical to our own.

□

4.2 An extension of $\mathbf{I}_2^{\mathrm{s}}$

A natural extension of $\mathbf{I}_2^{\mathrm{s}}$ is obtained by adding the rule of *subsumption* to the rules of $\mathbf{I}_2^{\mathrm{s}}$:

$$(\text{SUB}) \quad \frac{A \vdash M : \tau}{A \vdash M : \sigma} \quad (\text{where } \tau \leq_2 \sigma)$$

We write $\mathbf{I}_2 \triangleright A \vdash M : \sigma$ if the judgment $A \vdash M : \sigma$ follows by the rules of $\mathbf{I}_2^{\mathrm{s}}$ plus (SUB), with types appearing in type environments restricted to $\mathbf{T}_1$, and derived types restricted to $\mathbf{T}_2$.

Clearly, every judgment of $\mathbf{I}_2^{\mathrm{s}}$ is a judgment of $\mathbf{I}_2$. The converse does not hold; for example, the judgment

$$\{x : \sigma \to \tau\} \vdash x : (\sigma \wedge \sigma') \to \tau$$

is derivable in $\mathbf{I}_2$ for any $\sigma \neq \sigma' \in \mathbf{T}_0$, but is not derivable in $\mathbf{I}_2^{\mathrm{s}}$.

$\mathbf{I}_2$ has principal pairs, and indeed, they are identical to the principal pairs of $\mathbf{I}_2^{\mathrm{s}}$ (the proof is a simple extension of the proof of Theorem 41). An immediate consequence is that the terms typable in $\mathbf{I}_2$ are exactly the same as the terms typable in $\mathbf{I}_2^{\mathrm{s}}$.

In summary:

Theorem 43 (Comparison of $\mathbf{I}_2$ and $\mathbf{I}_2^{\mathrm{s}}$)

i) *If* $\mathbf{I}_2^{\mathrm{s}} \triangleright A \vdash M : \sigma$, *then* $\mathbf{I}_2 \triangleright A \vdash M : \sigma$. *The converse does not hold.*

ii) *A term M is typable in* $\mathbf{I}_2$ *if and only if it is typable in* $\mathbf{I}_2^{\mathrm{s}}$.

Although it does not type any more terms than $\mathbf{I}_2^{\mathrm{s}}$, $\mathbf{I}_2$ has other advantages over $\mathbf{I}_2^{\mathrm{s}}$.

Example 44 The acceptable pairs of $\mathbf{I}_2^{\mathrm{s}}$ are not closed under the operation $\leq$:

$$\mathbf{I}_2^{\mathrm{s}} \rhd \{x : s \to t\} \vdash x : s \to t,$$

and

$$\langle \{x : s \to t\}, s \to t \rangle \quad \leq \quad \langle \{x : s \to t\}, (s \wedge t) \to t \rangle,$$

but the judgment

$$\{x : s \to t\} \vdash x : (s \wedge t) \to t$$

is not derivable in $\mathbf{I}_2^{\mathrm{s}}$.

On the other hand, $\mathbf{I}_2$ is closed under $\leq$:

Lemma 45 (Weakening for $\mathbf{I}_2$) *If $\mathbf{I}_2 \rhd A \vdash M : \sigma$ and $\langle A, \sigma \rangle \leq \langle A', \sigma' \rangle$, then $\mathbf{I}_2 \rhd A' \vdash M : \sigma'$.*

For this reason, we prefer $\mathbf{I}_2$ to either $\mathbf{I}_2^{\mathrm{vb}}$ or $\mathbf{I}_2^{\mathrm{s}}$. However, it was still useful to develop $\mathbf{I}_2^{\mathrm{s}}$. In particular, the example above shows that Lemma 19 does not hold for $\mathbf{I}_2$; it was convenient to have Lemma 19 for the proof of the equivalence of typability with Λ_2^{s}.

5 Combining intersections and quantification

5.1 The system $\mathbf{P}_2$

We now describe a type system that combines aspects of rank 2 intersection types and rank 2 polymorphic types. The system is called $\mathbf{P}_2$, as it is the rank 2 subset of a type system $\mathbf{P}$ (described elsewhere).

The types of the system are the rank 2 intersection types extended with top-level quantifiers:

$$\mathbf{T}_{\forall 2} \quad = \quad \mathbf{T}_2 \cup \{(\forall t \sigma) \mid \sigma \in \mathbf{T}_{\forall 2}\}.$$

In order to simplify the definition of subtyping, we consider $\mathbf{T}_{\forall 2}$ types syntactically equal modulo renaming of bound type variables, reordering of adjacent quantifiers, and elimination of unnecessary quantifiers. When a $\mathbf{T}_{\forall 2}$ type is written in the form $\forall \vec{s} \sigma$, we assume $\sigma \in \mathbf{T}_2$.

Definition 46

i) The relation $\leq_{\forall 2}$ is the least partial order on $\mathbf{T}_{\forall 2}$ closed under the following rules:

a) If $\sigma \leq_2 \tau$, then $\sigma \leq_{\forall 2} \tau$.

b) If $\tau \in \mathbf{T}_0$, then $(\forall t\sigma) \leq_{\forall 2} \{t := \tau\}\sigma$.

c) If $\sigma \leq_{\forall 2} \tau$ and t is not free in σ, then $\sigma \leq_{\forall 2} (\forall t\tau)$.

ii) The relation $\leq_{\forall 2,1}$ between $\mathbf{T}_{\forall 2}$ and $\mathbf{T}_1$ is the least relation closed under the rule:

a) If $\sigma \leq_{\forall 2} \tau_i$ for all $i \in I$, then $\sigma \leq_{\forall 2,1} (\bigwedge_{i\in I} \tau_i)$.

The rules for $\leq_{\forall 2}$ express the intuition that a type is a subtype of its instances. They are equivalent to the following rule, similar to ML's notion of *generic instance*:

- If $\{\vec{s} := \vec{\rho}\}\sigma \leq_2 \tau$, where $\vec{\rho}$ is a vector of simple types, and the type variables $\vec{t}$ are not free in $(\forall\vec{s}\sigma)$, then $\forall\vec{s}\sigma \leq_{\forall 2} \forall\vec{t}\tau$.

Note that we only allow instantiation of simple types. This ensures that instantiation does not take us beyond rank 2. It also has less desirable implications, e.g., $(\forall t.t)$ is not a least type in the ordering $\leq_{\forall 2}$: $(\forall t.t) \not\leq_{\forall 2} (s_1 \wedge (s_1 \rightarrow s_2)) \rightarrow s_2$.

The relation $\leq_{\forall 2,1}$ is not a partial order; it is not even reflexive. This is because it relates types "across rank." Note that in a comparison

$$(\forall t\tau) \quad \leq_{\forall 2,1} \quad (\bigwedge_{i\in I} \sigma_i),$$

the variable t may be instantiated differently for each σ_i.

Some basic properties of $\leq_{\forall 2}$ and $\leq_{\forall 2,1}$ are summarized in the following lemma.

Lemma 47

i) *If $\sigma, \tau \in \mathbf{T}_0$, then $\sigma \leq_{\forall 2} \tau$ iff $\sigma \leq_{\forall 2,1} \tau$ iff $\sigma = \tau$.*

ii) *If $\sigma, \tau \in \mathbf{T}_2$, then $\sigma \leq_{\forall 2} \tau$ iff $\sigma \leq_2 \tau$.*

iii) *If $\sigma \leq_{\forall 2} \tau$, then $(\forall t\sigma) \leq_{\forall 2} (\forall t\tau)$.*

iv) *If $\sigma \in \mathbf{T}_2$ and $\tau \in \mathbf{T}_0$, then $\forall\vec{t}\sigma \leq_{\forall 2} \tau$ iff for some substitution S with $\mathbf{dom}(S) \subseteq \vec{t}$, we have $S\sigma = \tau$.*

v) *For any substitution S and types $\sigma, \tau \in \mathbf{T}_{\forall 2}$, if $S\sigma \leq_{\forall 2} \tau$, then $S(\forall t\sigma) \leq_{\forall 2} \tau$.*

vi) *For any substitution S, types $\sigma, \tau \in \mathbf{T}_{\forall 2}$, and type environment A, if $S\sigma \leq_{\forall 2} \tau$, then $S(\mathrm{Gen}(A, \sigma)) \leq_{\forall 2} \tau$.*

$$(\textsc{var}) \qquad \{x : (\bigwedge_{i\in I} \tau_i)\} \vdash x : \tau_{i_0} \quad (\text{where } i_0 \in I)$$

$$(\textsc{abs}) \qquad \frac{A \cup \{x : \sigma\} \vdash M : \tau}{A \vdash (\lambda x M) : \sigma \to \tau}$$

$$(\textsc{app}) \qquad \frac{A \vdash M : (\bigwedge_{i\in I} \tau_i) \to \sigma, \quad (\forall i \in I)\ A \vdash N : \tau_i}{A \vdash (MN) : \sigma}$$

$$(\textsc{gen}) \qquad \frac{A \vdash M : \sigma}{A \vdash M : \forall t \sigma} \quad (\text{where } t \notin \mathrm{FTV}(A))$$

$$(\textsc{sub}) \qquad \frac{A \vdash M : \tau}{A \vdash M : \sigma} \quad (\text{where } \tau \leq_{\forall 2} \sigma)$$

$$(\textsc{add-hyp}) \qquad \frac{A \vdash M : \sigma}{A \cup \{x : \tau\} \vdash M : \sigma}$$

Figure 7: Typing rules of $\mathbf{P}_2$. Types in type environments are in $\mathbf{T}_1$, and derived types are in $\mathbf{T}_{\forall 2}$.

vii) *If* $\sigma_1 \leq_{\forall 2} \sigma_2 \leq_{\forall 2,1} \sigma_3 \leq_1 \sigma_4$, *then* $\sigma_1 \leq_{\forall 2,1} \sigma_4$.

The typing rules of the system are presented in Figure 7. We write $\mathbf{P}_2 \triangleright A \vdash M : \sigma$ if the judgment $A \vdash M : \sigma$ follows by these rules, with types appearing in type environments restricted to $\mathbf{T}_1$, and derived types restricted to $\mathbf{T}_{\forall 2}$.

The rules differ from the rules of $\mathbf{I}_2$ in two respects. First, we have added the rule (GEN) so that we can derive quantified types, and $\leq_{\forall 2}$ is used in place of $\leq_2$ in the rule (SUB) so that the quantifiers can be instantiated. Second, we have added the rule (ADD-HYP), which was a derived rule in all of our previous type systems. The addition of (ADD-HYP) will simplify some subsequent definitions, and also allows us to weaken the rules (VAR) and (ABS) slightly. The old (VAR) and (ABS) are derivable in $\mathbf{P}_2$, so $\mathbf{P}_2$ extends $\mathbf{I}_2$:

Lemma 48 *If* $\mathbf{I}_2 \triangleright A \vdash M : \sigma$, *then* $\mathbf{P}_2 \triangleright A \vdash M : \sigma$.

In fact, a stronger connection can be shown: a term is typable in one system if and only if it is typable in the other.

Theorem 49 (Comparison of $\mathbf{P}_2$ and $\mathbf{I}_2$) *$\mathbf{P}_2 \triangleright A \vdash M : \forall\vec{t}\sigma$ for some $\vec{t}$ if and only if $\mathbf{I}_2 \triangleright A \vdash M : \sigma$.*

Proof: The right-to-left direction follows by Lemma 48. The left-to-right direction is proved by induction on derivations; the only non-trivial case is (SUB), which can be shown as follows.

If $\mathbf{P}_2 \triangleright A \vdash M : \forall\vec{t}\sigma$ follows by rule (SUB), then we must have a shorter derivation of

$$\mathbf{P}_2 \triangleright A \vdash M : \forall\vec{s}\tau,$$

and $(\forall\vec{s}\tau) \leq_{\forall 2} (\forall\vec{t}\sigma)$. We must show $\mathbf{I}_2 \triangleright A \vdash M : \sigma$.

By induction, $\mathbf{I}_2 \triangleright A \vdash M : \tau$. Furthermore, by the definition of $\leq_{\forall 2}$, for some sequence $\vec{\rho}$ of simple types, we have $\{\vec{s} := \vec{\rho}\}\tau \leq_2 \sigma$. We may assume that the type variables $\vec{s}$ do not appear in A. Then by substitutivity for $\mathbf{I}_2$,

$$\mathbf{I}_2 \triangleright A \vdash M : \{\vec{s} := \vec{\rho}\}\tau,$$

and by the $\mathbf{I}_2$ rule (SUB), we have $\mathbf{I}_2 \triangleright A \vdash M : \sigma$, as desired. $\square$

The ordering $\leq$ of Definition 28 is extended to pairs with $\mathbf{T}_{\forall 2}$ types as follows:

$$\langle A, \sigma\rangle \leq \langle A', \sigma'\rangle \text{ if and only if } A' \leq_1 A \text{ and } \sigma \leq_{\forall 2} \sigma'.$$

Lemma 50 (Weakening for $\mathbf{P}_2$) *If $\mathbf{P}_2 \triangleright A \vdash M : \sigma$ and $\langle A, \sigma\rangle \leq \langle A', \sigma'\rangle$, then $\mathbf{P}_2 \triangleright A' \vdash M : \sigma'$.*

Lemma 51 (Substitutivity for $\mathbf{P}_2$) *If $\mathbf{P}_2 \triangleright A \vdash M : \sigma$, then $\mathbf{P}_2 \triangleright SA \vdash M : S\sigma$ for any substitution S.*

5.2 Extending subtype satisfaction

In order to perform type inference for $\mathbf{P}_2$, we will need to solve problems that generalize the $\leq_{2,1}$-satisfaction problems of §3.1.

A $\leq_{\forall 2,1}$-*satisfaction problem* π is a pair $\exists\vec{s}.P$, where P is a finite set whose every element is either: 1) an equality between simple types; or 2) an inequality between a $\mathbf{T}_{\forall 2}$ type and a $\mathbf{T}_1$ type. A substitution S is a *solution* to $\exists\vec{s}.P$ if there is a substitution S' such that $S(t) = S'(t)$ for all $t \notin \vec{s}$, $S'\sigma \leq_{\forall 2,1} S'\tau$ for all inequalities $(\sigma \leq \tau) \in P$, and $S'\sigma = S'\tau$ for all equalities $(\sigma = \tau) \in P$.

Note that any $\leq_{2,1}$-satisfaction problem is a $\leq_{\forall 2,1}$-satisfaction problem with the same set of solutions. Therefore we abuse notation and write

Solutions(π), **MGS**(π), and **MGS**$(\pi)[W]$ for the solutions, most general solutions, and most general solutions away from W of a $\leq_{\forall 2,1}$-satisfaction problem π.

Similarly, $\leq_{\forall 2,1}$-satisfaction problems can be solved by extending the transformational algorithm of Figure 6 by the following rule:

$$(\forall t \sigma) \leq \tau \quad \Rightarrow \quad \exists t\{\sigma \leq \tau\}$$
$$\text{if } \tau \text{ is not a } \wedge\text{-type, and } t \text{ is not free in } \tau$$

Theorem 52 *Every $\leq_{\forall 2,1}$-satisfaction problem is equivalent to a unification problem, and moreover, there is an algorithm that transforms every $\leq_{\forall 2,1}$-satisfaction problem into an equivalent unification problem.*

Proof: We show that the rules of Figure 6, augmented by the rule above, constitute an algorithm for converting any $\leq_{\forall 2,1}$-satisfaction problem into an equivalent unification problem (equalities are between simple types, inequalities are between $\mathbf{T}_{\forall 2}$ and $\mathbf{T}_1$ types).

First, note that every rule transforms a $\leq_{\forall 2,1}$-satisfaction problem into another $\leq_{\forall 2,1}$-satisfaction problem.

Second, note that each rule preserves the set of solutions, so that each application of a rule transforms a problem into an equivalent problem.

Third, note that repeated application of these rules must halt: every rule reduces the number of type constructors ('$\rightarrow$', '$\wedge$', or '$\forall$') in inequalities or reduces the number of inequalities.

Finally, note that a normal form contains no inequalities, and is therefore a unification problem. □

Corollary 53 *Let π be a $\leq_{\forall 2,1}$-satisfaction problem and W be a finite set of type variables.*

i) **Solutions**$(\pi) = \emptyset$ *iff* **MGS**$(\pi) = \emptyset$ *iff* **MGS**$(\pi)[W] = \emptyset$.

ii) *There is an algorithm that decides whether π has a solution, and, if so, returns an element of* **MGS**$(\pi)[W]$.

Theorem 54 *The subtyping relation $\leq_{\forall 2,1}$ is decidable.*

Proof: To see whether $\sigma \leq_{\forall 2,1} \tau$, compute $U \in$ **MGS**$(\{\sigma \leq \tau\})$ and check to see whether U is the identity substitution. □

5.3 Type inference for $\mathbf{P}_2$

Definition 55 For any term M, we define the set $\mathrm{PP}_{\mathbf{P}_2}(M)$ by induction on M.

i) If $M = x$, then $\langle \{x : t\}, t\rangle \in \mathrm{PP}_{\mathbf{P}_2}(x)$ for any type variable t.

ii) If $M = \lambda x N$, and $\langle A, \forall \vec{s}\sigma\rangle \in \mathrm{PP}_{\mathbf{P}_2}(N)$, where the type variables $\vec{s}$ are distinct from all other type variables, then:

   a) If $x \notin \mathbf{dom}(A)$, and t is a fresh type variable, then

   $$\langle A, \forall t\vec{s}(t \to \sigma)\rangle \in \mathrm{PP}_{\mathbf{P}_2}(\lambda x N).$$

   b) If $x \in \mathbf{dom}(A)$, then $\langle A_x, \mathrm{Gen}(A_x, A(x) \to \sigma)\rangle \in \mathrm{PP}_{\mathbf{P}_2}(\lambda x N)$.

iii) If $M = M_1 M_2$, and $\langle A_1, \forall \vec{s}\sigma_1\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_1)$, then:

   a) If $\sigma_1 = t$ (a type variable), t_1 and t_2 are fresh type variables, the type variables of $\langle A_2, \sigma_2\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_2)$ are fresh, $U \in \mathbf{MGS}(\{\sigma_2 \leq t_1, t = t_1 \to t_2\})$, and $A = U(A_1 + A_2)$, then

   $$\langle A, \mathrm{Gen}(A, Ut_2)\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M).$$

   b) If $\sigma_1 = (\bigwedge_{i \in I} \tau_i) \to \tau$, $(\forall i \in I)$ the type variables of $\langle A_i, \sigma_i\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_2)$ are fresh, $U \in \mathbf{MGS}(\{\sigma_i \leq \tau_i \mid i \in I\})$, and $A = U(A_1 + \sum_{i \in I} A_i)$, then

   $$\langle A, \mathrm{Gen}(A, U\tau)\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M).$$

Just as with $\mathbf{I}_2^{\mathrm{s}}$, the elements of $\mathrm{PP}_{\mathbf{P}_2}(M)$ are trivial variants of each other, so Definition 55 can easily be adapted to a type inference algorithm.

Lemma 56

i) *If $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$, then $x \in \mathbf{dom}(A)$ if and only if x is free in M.*

ii) *Suppose $\langle A_1, \sigma_1\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$. Then $\langle A_2, \sigma_2\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$ if and only if there is a bijection R of type variables such that $R\langle A_1, \sigma_1\rangle = \langle A_2, \sigma_2\rangle$.*

Proof: An easy induction on Definition 55. □

Theorem 57 *There is an algorithm that decides, for any M, whether the set $\mathrm{PP}_{\mathbf{P}_2}(M)$ is empty; and furthermore, if $\mathrm{PP}_{\mathbf{P}_2}(M)$ is not empty, it produces a member of $\mathrm{PP}_{\mathbf{P}_2}(M)$.*

Proof: Just follow the rules of Definition 55, generating "fresh" type variables as necessary, and use the algorithm of Corollary 53 to compute **MGS**. □

Theorem 58 *If* $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$, *then* $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$.

Proof: By induction on the definition of $\mathrm{PP}_{\mathbf{P}_2}(M)$.

i) If $M = x$, then $\langle A, \sigma\rangle = \langle \{x : t\}, t\rangle$ for some type variable t.
   Then we have $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(x)$ by rule (VAR).

ii) If $M = \lambda x N$, then by Lemma 56(i) we must consider two cases:

   a) It x is not free in N, then $\langle A, \forall\vec{s}\sigma'\rangle \in \mathrm{PP}_{\mathbf{P}_2}(N)$ for some σ', and $\sigma = \forall t\vec{s}(t \to \sigma')$ for some fresh type variable t.
   By induction, $\langle A, \forall\vec{s}\sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2}(N)$, and by weakening,

   $$\langle A \cup \{x : t\}, \sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2}(N).$$

   Note that $A \cup \{x : t\}$ is well-formed by Lemma 56(i).
   By rule (ABS), $\langle A, t \to \sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2}(\lambda x N)$, and by (GEN),

   $$\langle A, \forall t\vec{s}(t \to \sigma')\rangle = \langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(\lambda x N).$$

   b) If x is free in N, then $\langle A, \sigma\rangle = \langle A'_x, \mathrm{Gen}(A'_x, A'(x) \to \sigma')\rangle$, where $\langle A', \forall\vec{s}\sigma'\rangle \in \mathrm{PP}_{\mathbf{P}_2}(N)$.
   By induction and rule (SUB), $\langle A', \sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2}(N)$, so by rule (ABS), $\langle A'_x, A'(x) \to \sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2}(\lambda x N)$. Then by (GEN),

   $$\langle A'_x, \mathrm{Gen}(A'_x, A'(x) \to \sigma')\rangle = \langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(\lambda x N).$$

iii) If $M = M_1 M_2$, then we have $\langle A_1, \forall\vec{s}\sigma_1\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_1)$. By induction, $\langle A_1, \forall\vec{s}\sigma_1\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1)$, and by (SUB), $\langle A_1, \sigma_1\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1)$.

   a) If σ_1 is a type variable t, then we must have a pair $\langle A_2, \sigma_2\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_2)$ with fresh type variables, $A = U(A_1 + A_2)$ and $\sigma = \mathrm{Gen}(A, Ut_2)$, where $U \in \mathbf{MGS}(\{\sigma_2 \le t_1, t = t_1 \to t_2\})$ for fresh type variables t_1 and t_2.
   By induction, $\langle A_2, \sigma_2\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_2)$. By substitutivity,

   $$U\langle A_1, \sigma_1\rangle = \langle UA_1, (Ut_1) \to (Ut_2)\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1)$$

and

$$U\langle A_2, \sigma_2\rangle = \langle UA_2, U\sigma_2\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_2).$$

By weakening,

$$\langle UA_1 + UA_2, (Ut_1) \to (Ut_2)\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1)$$

and

$$\langle UA_1 + UA_2, Ut_1\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_2).$$

Then by rule (APP) we have

$$\langle UA_1 + UA_2, Ut_2\rangle = \langle A, Ut_2\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1M_2),$$

and by rule (GEN),

$$\langle A, \mathrm{Gen}(A, Ut_2)\rangle = \langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1M_2).$$

b) If $\sigma_1 = (\bigwedge_{i\in I} \tau_i) \to \tau$, then we must have pairs $\langle A_i, \sigma_i\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_2)$ with fresh type variables, $U \in \mathbf{MGS}\{\sigma_i \leq \tau_i \mid i \in I\}$, $A = U(A_1 + \sum_{i\in I} A_i)$, and $\sigma = \mathrm{Gen}(A, U\tau)$.
By induction and substitutivity, $\langle UA_i, U\sigma_i\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_2)$ for all $i \in I$, and by substitutivity we have

$$\langle UA_1, U\sigma_1\rangle = \langle UA_1, (\bigwedge_{i\in I} U\tau_i) \to U\tau\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1).$$

By weakening, $\langle A, (\bigwedge_{i\in I} U\tau_i) \to U\tau\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1)$ and $\langle A, U\tau_i\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_2)$ for all $i \in I$.
Then by rules (APP) and (GEN) we have

$$\langle A, \mathrm{Gen}(UA, U\tau)\rangle = \langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M).$$

□

Theorem 59 (Principal pairs for $\mathbf{P}_2$) *If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$, then there is a pair $\langle A', \sigma'\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$ and a substitution S such that $S\langle A', \sigma'\rangle \leq \langle A, \sigma\rangle$.*

Proof: By induction on the definition of $\mathrm{AP}_{\mathbf{P}_2}(M)$.

i) If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$ by rule (VAR), then $M = x$ for some variable x, $A(x) = (\bigwedge_{i\in I} \sigma_i)$, and $\sigma = \sigma_{i_0} \in \mathbf{T}_0$ for some $i_0 \in I$.

By the definition of $\mathrm{PP}_{\mathbf{P}_2}$, $\langle \{x : t\}, t\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$, where t is a fresh type variable.

Then $\{t := \sigma\}$ is a well-formed substitution and

$$\{t := \sigma\}\langle \{x : t\}, t\rangle = \langle \{x : \sigma\}, \sigma\rangle \leq \langle A, \sigma\rangle.$$

ii) If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$ by rule (ABS), then $M = \lambda x N$, σ is of the form $\sigma_1 \to \sigma_2$, and $\langle A \cup \{x : \sigma_1\}, \sigma_2\rangle \in \mathrm{AP}_{\mathbf{P}_2}(N)$.

By induction, there is a substitution S' and pair $\langle A', \forall\vec{s}\sigma_2'\rangle \in \mathrm{PP}_{\mathbf{P}_2}(N)$ such that

$$S'\langle A', \forall\vec{s}\sigma_2'\rangle \leq \langle A \cup \{x : \sigma_1\}, \sigma_2\rangle. \tag{2}$$

a) If $x \notin \mathbf{dom}(A')$, then for any fresh type variable t,

$$\langle A', \forall t\vec{s}(t \to \sigma_2')\rangle \in \mathrm{PP}_{\mathbf{P}_2}(\lambda x N).$$

It remains to show that there is a substitution S such that

$$S\langle A', \forall t\vec{s}(t \to \sigma_2')\rangle \leq \langle A, \sigma\rangle.$$

Just let $S = S'$. By (2), we have $A \leq_1 S'A'$, so we only need show

$$S'(\forall t\vec{s}(t \to \sigma_2')) \leq_{\forall 2} \sigma_1 \to \sigma_2.$$

We can assume $t, \vec{s}$ are fresh, so that

$$S'(\forall t\vec{s}(t \to \sigma_2')) = \forall t\vec{s}(t \to S'\sigma_2').$$

And by (2),

$$\{t := \sigma_1\}\forall\vec{s}(t \to S'\sigma_2') = \forall\vec{s}(\sigma_1 \to S'\sigma_2') \leq_{\forall 2} \sigma_1 \to \sigma_2,$$

so by the definition of $\leq_{\forall 2}$, $S'(\forall t\vec{s}(t \to \sigma_2')) \leq_{\forall 2} \sigma_1 \to \sigma_2$ as desired.

b) If $x \in \mathbf{dom}(A')$, then $\langle A'_x, \mathrm{Gen}(A'_x, A'(x) \to \sigma_2')\rangle \in \mathrm{PP}_{\mathbf{P}_2}(\lambda x N)$. Then by (2) and the definition of $\leq$,

$$S'\langle A'_x, \mathrm{Gen}(A'_x, A'(x) \to \sigma_2')\rangle \leq \langle A, \sigma_1 \to \sigma_2\rangle,$$

as desired.

iii) If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$ by rule (APP), then $M = M_1M_2$, and we have $\langle A, (\bigwedge_{i\in I}\sigma_i) \to \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_1)$ and $\langle A, \sigma_i\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M_2)$ for all $i \in I$.

By induction, $\mathrm{PP}_{\mathbf{P}_2}(M_1)$ is nonempty, and by Lemma 56(ii), it is sufficient to consider the following cases on the structure of pairs in $\mathrm{PP}_{\mathbf{P}_2}(M_1)$.

a) $\langle A_1, \forall\vec{s}t\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_1)$.
We assume that the type variables $\vec{s}$ are fresh, so by induction and the definition of $\leq_{\forall 2}$, there is a substitution S_1 such that

$$S_1\langle A_1, t\rangle \quad\leq\quad \langle A, (\bigwedge_{i\in I}\sigma_i) \to \sigma\rangle.$$

By the definition of $\leq_{\forall 2}$, $S_1t = (\sigma_{i_0} \to \sigma')$ for some $i_0 \in I$ and $\sigma' \in \mathbf{T}_0$ with $\sigma' \leq_{\forall 2} \sigma$.
By induction and Lemma 56(ii), there is a disjoint pair $\langle A_2, \tau\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_2)$ and substitution S_2 such that

$$S_2\langle A_2, \tau\rangle \quad\leq\quad \langle A, \sigma_{i_0}\rangle.$$

Let $\pi = \{\ \tau \leq t_1, t = t_1 \to t_2\ \}$, where t_1, t_2 are fresh. Note that π has a solution, $S = S_1 \cup S_2 \cup \{t_1 := \sigma_{i_0}, t_2 := \sigma'\}$. Therefore, we may pick $U \in \mathbf{MGS}(\pi)$ and let $A' = U(A_1 + A_2)$, so that

$$\langle A', \mathrm{Gen}(A', Ut_2)\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_1M_2).$$

By Convention 35, there exists an R such that $RUt_2 = St_2 = \sigma' \leq_{\forall 2} \sigma$, and

$$A \leq_1 S_1A_1 + S_2A_2 = S(A_1 + A_2) = RU(A_1 + A_2) = RA'.$$

By Lemma 47(vi), $R(\mathrm{Gen}(Ut_2)) \leq_{\forall 2} \sigma$, so

$$R\langle A', \mathrm{Gen}(A', Ut_2)\rangle \leq \langle A, \sigma\rangle,$$

as desired.

b) $\langle A_1, \forall\vec{s}(\bigwedge_{j\in J}\sigma'_j) \to \sigma'\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_1)$.
We assume that the type variables $\vec{s}$ are fresh, so by induction and the definition of $\leq_{\forall 2}$, there is a substitution S_1 such that

$$S_1\langle A_1, (\bigwedge_{j\in J}\sigma'_j) \to \sigma'\rangle \quad\leq\quad \langle A, (\bigwedge_{i\in I}\sigma_i) \to \sigma\rangle.$$

Then $\{S_1\sigma'_j \mid j \in J\} \subseteq \{\sigma_i \mid i \in I\}$, so for all $j \in J$ there is an $i_j \in I$ such that $S_1\sigma'_j = \sigma_{i_j}$.

By induction and Lemma 56(ii), for all $j \in J$ there are disjoint pairs $\langle A_j, \rho_j\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_2)$ and substitutions S_j such that

$$S_j\langle A_j, \rho_j\rangle \quad \leq \quad \langle A, \sigma_{i_j}\rangle.$$

Let $\pi = \{\rho_j \leq \sigma'_j \mid j \in J\}$. Then $S = S_1 \cup (\bigcup_{j\in J} S_j)$ is a solution to π: $S\rho_j = S_j\rho_j \leq_2 \sigma_{i_j} = S_1\sigma'_j = S\sigma'_j$.

Pick $U \in \mathbf{MGS}(\pi)$, and let $A' = U(A_1 + \sum_{j\in J} A_j)$. Then

$$\langle A', \mathrm{Gen}(A', U\sigma')\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M_1M_2).$$

By Convention 35, there exists an R such that $RU\sigma' = S\sigma' = S_1\sigma' \leq_{\forall 2} \sigma$, and

$$A \leq_1 S_1A_1 + \sum_{j\in J} S_jA_j = S(A_1 + \sum_{j\in J} A_j) = RU(A_1 + \sum_{j\in J} A_j) = RA'.$$

By Lemma 47(vi), $R(\mathrm{Gen}(U\sigma')) \leq_{\forall 2} \sigma$, so

$$R\langle A', \mathrm{Gen}(A', U\sigma')\rangle \leq \langle A, \sigma\rangle,$$

as desired.

iv) If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$ by rule (GEN), then $\sigma = \forall t\sigma'$, we have a shorter derivation of $\langle A, \sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$, and $t \notin \mathrm{FTV}(A)$.

By induction there is a pair $\langle A', \sigma''\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$ and a substitution S such that $S\langle A', \sigma''\rangle \leq \langle A, \sigma'\rangle$.

We now show that $t \notin \mathrm{FTV}(S\sigma'')$. Then since $S\sigma'' \leq_{\forall 2} \sigma'$, we have $S\sigma'' \leq_{\forall 2} \forall t\sigma' = \sigma$, as desired.

Assume by way of contradiction that $t \in \mathrm{FTV}(S\sigma'')$. Since $A \leq_1 SA'$, $\mathrm{FTV}(SA') \subseteq \mathrm{FTV}(A)$. Therefore, $t \notin \mathrm{FTV}(A) \Rightarrow t \notin \mathrm{FTV}(SA')$.

Since $t \notin \mathrm{FTV}(SA')$ and $t \in \mathrm{FTV}(S\sigma'')$, there must be some $u \in \mathrm{FTV}(\sigma'') - \mathrm{FTV}(A')$ such that $t \in \mathrm{FTV}(Su)$. However, it is easily checked that $\langle A', \sigma''\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M) \Rightarrow \mathrm{FTV}(\sigma'') - \mathrm{FTV}(A') = \emptyset$, so we have reached a contradiction.

v) If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$ by rule (SUB), then for some $\sigma' \leq_{\forall 2} \sigma$, we have a shorter derivation of $\langle A, \sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$.

By induction there is a pair $\langle A', \sigma''\rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$ and a substitution S such that $S\langle A', \sigma''\rangle \leq \langle A, \sigma'\rangle$.

Then by transitivity, $S\sigma'' \leq_{\forall 2} \sigma$ as desired.

vi) If $\langle A, \sigma \rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$ by rule (ADD-HYP), then $A = A_x \cup \{x : \tau\}$ and $\langle A_x, \sigma \rangle \in \mathrm{AP}_{\mathbf{P}_2}(M)$.

By induction there is a pair $\langle A', \sigma' \rangle \in \mathrm{PP}_{\mathbf{P}_2}(M)$ and a substitution S such that $S\langle A', \sigma' \rangle \leq \langle A_x, \sigma \rangle$.

Then $S\langle A', \sigma' \rangle \leq \langle A, \sigma \rangle$ as desired.

□

6 Recursive definitions

We now consider ways of typing recursive definitions. We extend the grammar of our language to include terms of the form $(\mu x M)$. Such a term is meant to represent the recursive program x such that $x = M$ (M may contain occurrences of x).

In ML, recursive definitions are typed by the following rule:

$$(\textsc{rec-simple}) \quad \frac{A_x \cup \{x : \tau\} \vdash M : \tau}{A \vdash (\mu x M) : \tau} \quad (\text{where } \tau \in \mathbf{T}_0)$$

6.1 Recursive definitions in Λ_2

In Λ_2 and ML, the rule (REC-SIMPLE) seems overly restrictive. Both systems allow ML type schemes to appear in type environments and as derived types, suggesting the rule of *polymorphic recursion*:

$$(\textsc{rec-poly}) \quad \frac{A_x \cup \{x : \tau\} \vdash M : \tau}{A \vdash (\mu x M) : \tau} \quad (\text{where } \tau \in \mathbf{S}(1))$$

Example 60 When extended by (REC-POLY), both ML and Λ_2 can type the following terms:

$$\begin{aligned}
(\mu w.(\lambda xy.y)(ww)) &: \forall t.t \to t, \\
(\mu w.(\lambda xyz.z)(w\,3)(w\,\mathbf{true})) &: \forall t.t \to t, \\
(\mu x.xx) &: \forall t.t.
\end{aligned}$$

Neither is typable with the rule (REC-SIMPLE). Other examples are given by Mycroft [25] and Kfoury et al. [13, 15], who introduced (REC-POLY) independently.

Unfortunately, type inference for Λ_2 or ML extended by (REC-POLY) is undecidable [14, 9], so (REC-SIMPLE) is used in practice.

6.2 Recursive definitions in $\mathbf{I}_2$

The rule (REC-SIMPLE) is one way of typing recursive definitions in intersection type systems. However, as with ML and Λ_2, it seems overly restrictive. The rule (REC-POLY) involves $\mathbf{S}(1)$ types, so it is not appropriate for the intersection type systems. Instead, we might consider a rule like the following:

$$\frac{A_x \cup \{x : \tau)\} \vdash M : \tau}{A \vdash (\mu x M) : \tau} \quad (\text{where } \tau \in \mathbf{T}_1)$$

Note that the full power of the rule is achieved only by allowing $\mathbf{T}_1$ derived types, so the rule is not compatible with the rank 2 intersection type systems that we have defined so far. However, the rule can be adapted to our systems as follows:

$$(\text{REC-INT}) \quad \frac{(\forall i \in I)\; A_x \cup \{x : (\bigwedge_{j \in I} \tau_j)\} \vdash M : \tau_i}{A \vdash (\mu x M) : \tau_{i_0}} \quad (\text{where } i_0 \in I)$$

Example 61 The system $\mathbf{I}_2 + (\text{REC-INT})$ can type the following terms:

$$\begin{aligned} (\mu w.(\lambda xy.y)(ww)) &\;:\; \tau \to \tau, \\ (\mu w.(\lambda xyz.z)(w\,3)(w\,\mathbf{true})) &\;:\; \tau \to \tau, \end{aligned}$$

where τ is any simple type. Neither term is typable in $\mathbf{I}_2 + (\text{REC-SIMPLE})$.

The close connection between $\mathbf{I}_2$ and Λ_2 casts some doubt on the decidability of the system $\mathbf{I}_2 + (\text{REC-INT})$. However, $\mathbf{I}_2 + (\text{REC-INT})$ cannot type all of the terms that can be typed by $\Lambda_2 + (\text{REC-POLY})$. For example, the term $(\mu x.xx)$ cannot be typed in $\mathbf{I}_2 + (\text{REC-INT})$. The decidability of $\mathbf{I}_2 + (\text{REC-INT})$ is an open question.

6.3 Recursive definitions in $\mathbf{P}_2$

The system $\mathbf{P}_2$ could be extended to type recursive definitions with either the rule (REC-SIMPLE) or the rule (REC-INT) (the rule (REC-POLY) is not appropriate since it requires $\mathbf{S}(1)$ types to appear in type environments). Surprisingly, however, we can do better: by using the rule (REC) below, we will be able to type more terms than (REC-SIMPLE), while retaining principal typings and decidable type inference.

$$(\text{REC}) \qquad \frac{A \cup \{x : \tau\} \vdash M : \sigma}{A \vdash (\mu x M) : \sigma} \qquad (\text{where } \sigma \leq_{\forall 2,1} \tau)$$

Example 62

i) The following terms are typable in $\mathbf{P}_2$+(REC), but are not typable in $\mathbf{P}_2$ + (REC-SIMPLE):

$$\begin{aligned}(\mu w.(\lambda xy.y)(ww)) &: \forall t.t \to t,\\ (\mu w.(\lambda xyz.z)(w\,3)(w\,\mathbf{true})) &: \forall t.t \to t.\end{aligned}$$

ii) The term $(\mu x.xx)$ is not typable in $\mathbf{P}_2$+(REC). It has type $(\forall t.t)$ in ML+(REC-POLY) and Λ_2+(REC-POLY).

There is an anomaly in the rule (REC): if x does not appear in M, and M does not have a simple type, then $(\mu x M)$ will not be typable with (REC). For example, the term $(\mu x(\lambda y.yy))$ is not typable because $(\lambda y.yy)$ is only typable at rank 2; there is no $\mathbf{T}_1$ type that could be assigned to x so that (REC) applies.

This could be repaired by adding a special rule for the vacuous case:

$$(\text{REC-VAC}) \qquad \frac{A \vdash M : \sigma}{A \vdash (\mu x M) : \sigma} \qquad (\text{where } x \notin \mathbf{dom}(A))$$

Using rule (REC-VAC), we may derive the typing

$$(\mu w(\lambda x.xx)) : \forall s,t.(s \wedge (s \to t)) \to t.$$

However, to simplify our definitions we will not consider (REC-VAC) further.

6.4 Mutual recursion in $\mathbf{P}_2$

We now extend the grammar of our language to include terms of the form (**letrec** B **in** N), where B is a set of mutually recursive definitions. A particular B may be written as $x_1 = M_1, x_2 = M_2, \ldots, x_n = M_n$ or $\{x_i = M_i \mid i \in I\}$, where all of the variables x_i are distinct.

Figure 8 gives three rules for typing mutually recursive definitions. The first rule, (LETREC-SIMPLE), is used for typing mutual recursion in ML. Notice that in (LETREC-SIMPLE), the recursive definitions must be typed under the assumption that the recursive variables have simple type. In typing the body of the definition, however, the types of the recursive variables can be generalized.

We cannot use (LETREC-SIMPLE) with $\mathbf{P}_2$, because $\mathbf{P}_2$ does not permit quantified types to appear in type environments. And it is not easily adapted to $\mathbf{P}_2$. In ML, the polymorphic type $\text{Gen}(A, \tau_i)$ of x_i is easily obtained from the simple type τ_i used in typing the recursive definitions. The equivalent of

$$
(\text{LETREC-SIMPLE}) \quad \frac{\begin{array}{c}(\forall j \in I) \quad A \cup \{x_i : \tau_i \mid i \in I\} \vdash M_j : \tau_j \quad (\tau_j \in \mathbf{T}_0) \\ A \cup \{x_i : \mathrm{Gen}(A, \tau_i) \mid i \in I\} \vdash M : \sigma\end{array}}{A \vdash (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ M) : \sigma}
$$

$$
(\text{LETREC-VAR}) \quad \frac{\forall j \in I \quad A \cup \{x_i : \tau_i \mid i \in I\} \vdash M_j : \sigma_j \quad (\sigma_j \leq_{\forall 2,1} \tau_j)}{A \vdash (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ x_{i_0}) : \sigma_{i_0}} \quad i_0 \in I
$$

$$
(\text{LETREC}) \quad \frac{\begin{array}{c}A \cup \{x_i : \tau_i \mid i \in I\} \vdash N : \sigma \\ \forall j \in I \quad A \vdash_{\wedge} (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ x_j) : \tau_j\end{array}}{A \vdash (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ N) : \sigma} \quad N \notin \{x_i \mid i \in I\}
$$

Figure 8: Rules for typing recursive definitions

$\mathrm{Gen}(A, \tau_i)$ in $\mathbf{P}_2$ is some intersection $(\bigwedge_{j \in J} \tau_j)$, where each τ_j is an instance of $\mathrm{Gen}(A, \tau_i)$. It is not immediately clear how to get directly from τ_i to $(\bigwedge_{j \in J} \tau_j)$.

Instead, we use the following property of (LETREC-SIMPLE) as a guide in formulating the rules for $\mathbf{P}_2$.

Definition 63 If $B = \{x_1 = M_1, x_2 = M_2, \ldots, x_n = M_n\}$, then we define $\langle\!\langle \mathbf{letrec}\ B\ \mathbf{in}\ N \rangle\!\rangle$ to be the term

$$
\begin{array}{rl}
(\mathbf{let} & x_1 = (\mathbf{letrec}\ B\ \mathbf{in}\ x_1), \\
& \vdots \\
& x_n = (\mathbf{letrec}\ B\ \mathbf{in}\ x_n) \\
\mathbf{in} & N).
\end{array}
$$

Lemma 64 *If $M = (\mathbf{letrec}\ B\ \mathbf{in}\ N)$, then in* ML + (LETREC-SIMPLE), *$A \vdash M : \sigma$ iff $A \vdash \langle\!\langle M \rangle\!\rangle : \sigma$.*

The lemma shows how to type any term $(\mathbf{letrec}\ B\ \mathbf{in}\ N)$ given only typings for terms $(\mathbf{letrec}\ B\ \mathbf{in}\ x)$, where x is defined in B. This is the intuition behind the rules (LETREC-VAR) and (LETREC) of Figure 8. (LETREC-VAR) is a straightforward generalization of the rule (REC) for terms of the form $(\mathbf{letrec}\ B\ \mathbf{in}\ x)$, where x is a variable defined in B. The above lemma suggests that we type other **letrec** expressions by a rule of the form

$$
\frac{A \vdash \langle\!\langle \mathbf{letrec}\ B\ \mathbf{in}\ N \rangle\!\rangle : \sigma}{A \vdash (\mathbf{letrec}\ B\ \mathbf{in}\ N) : \sigma} \quad (N \text{ is not defined by } B)
$$

Of course, **let** expressions are not defined in $\mathbf{P}_2$. Therefore, in any language except ML, we will regard $(\mathbf{let}\ x = M\ \mathbf{in}\ N)$ as syntactic sugar for $(\lambda xN)M$, and $(\mathbf{let}\ x_1 = M_1, x_2 = M_2\ \mathbf{in}\ N)$ as syntactic sugar for $(\mathbf{let}\ x_1 = M_1\ \mathbf{in}\ (\mathbf{let}\ x_2 = M_2\ \mathbf{in}\ N))$.

Our rule (LETREC) is obtained simply by desugaring the **let** expression $\langle\!\langle M \rangle\!\rangle$ into abstractions and applications, and considering how the resulting term would be typed by (ABS) and (APP). We make the rule more compact by using the notation $A \vdash_\wedge M : (\bigwedge_{i\in I} \tau_i)$ to abbreviate $(\forall i \in I)\ A \vdash M : \tau_i$.

We write $\mathbf{P}_2^{\mathrm{R}} \rhd A \vdash M : \sigma$ if the judgment $A \vdash M : \sigma$ follows by the rules of $\mathbf{P}_2$ and the rules (REC), (LETREC-VAR), and (LETREC), with types appearing in type environments restricted to $\mathbf{T}_1$, and derived types restricted to $\mathbf{T}_{\forall 2}$.

Lemma 65 (Weakening for $\mathbf{P}_2^{\mathrm{R}}$) *If $\mathbf{P}_2^{\mathrm{R}} \rhd A \vdash M : \sigma$ and $\langle A, \sigma\rangle \leq \langle A', \sigma'\rangle$, then $\mathbf{P}_2^{\mathrm{R}} \rhd A' \vdash M : \sigma'$.*

Lemma 66 (Substitutivity for $\mathbf{P}_2^{\mathrm{R}}$) *If $\mathbf{P}_2^{\mathrm{R}} \rhd A \vdash M : \sigma$, then $\mathbf{P}_2^{\mathrm{R}} \rhd SA \vdash M : S\sigma$ for any substitution S.*

Lemma 67 *If $M = (\mathbf{letrec}\ B\ \mathbf{in}\ N)$, then $\mathbf{P}_2^{\mathrm{R}} \rhd A \vdash M : \sigma$ iff $\mathbf{P}_2^{\mathrm{R}} \rhd A \vdash \langle\!\langle M \rangle\!\rangle : \sigma$.*

6.5 Type inference in $\mathbf{P}_2^{\mathrm{R}}$

We define the type inference algorithm for $\mathbf{P}_2^{\mathrm{R}}$, and show that it is sound and complete.

Definition 68 The set $\mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$ of principal pairs for a term M is defined by extending the definition of $\mathrm{PP}_{\mathbf{P}_2}$ (Definition 55) by the following cases.

iv) If $M = (\mu x N)$ and $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$, then:

  a) If $x \notin \mathbf{dom}(A)$, and $U \in \mathbf{MGS}(\{\sigma \leq t\})$ where t is a fresh type variable,
  then $\langle UA, \mathrm{Gen}(UA, U\sigma)\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$.

  b) If $x \in \mathbf{dom}(A)$ and $U \in \mathbf{MGS}(\{\sigma \leq A(x)\})$,
  then $\langle UA_x, \mathrm{Gen}(UA_x, U\sigma)\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$.

v) If $M = (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ x_{i_0})$, where $i_0 \in I$,
and $\langle A_i, \sigma_i\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M_i)$ for $i \in I$,
  $A' = \sum_{i\in I} A_i$,
  $A'' = A' \cup \{x_i : t_i \mid i \in I,\ x_i \notin \mathbf{dom}(A'),\ t_i \text{ fresh}\}$,
  $U \in \mathbf{MGS}(\{\sigma_i \leq A''(x_i) \mid i \in I\})$,
  and $A = UA''_{\{x_i \mid i\in I\}}$,

then $\langle A, \mathrm{Gen}(A, U\sigma_{i_0})\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$.

vi) If $M = (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ N)$, where $N \notin \{x_i \mid i \in I\}$, and $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(\langle\!\langle M\rangle\!\rangle)$,
then $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$.

Theorem 69 *If $\langle A, \sigma\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$, then $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$.*

Proof: By induction on the definition of $\mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$. For the rules of $\mathbf{P}_2$, see the proof of Theorem 58. We only need to consider the following cases.

iv) If $M = (\mu x N)$, we consider two cases.

a) If x is not free in N, then for some $\langle A', \sigma'\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$, fresh type variable t, and $U \in \mathbf{MGS}(\{\sigma' \le t\})$,

$$\langle A, \sigma\rangle = \langle UA', \mathrm{Gen}(UA', U\sigma')\rangle.$$

By induction and substitutivity, $\langle UA', U\sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$. By weakening, $\langle UA' \cup \{x : Ut\}, U\sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$. And by the rules (REC) and (GEN),

$$\langle UA', \mathrm{Gen}(UA', U\sigma')\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(\mu x N),$$

as desired.

b) If x is free in N, then for some $\langle A', \sigma'\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$ and $U \in \mathbf{MGS}(\sigma' \le A'(x))$, we have

$$\langle A, \sigma\rangle = \langle UA'_x, \mathrm{Gen}(UA'_x, U\sigma')\rangle.$$

By induction, $\langle A', \sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$. Then $\langle UA', U\sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$ by substitutivity. Since $U\sigma' \le_{\forall 2,1} UA'(x)$, by rule (REC) we have $\langle UA'_x, U\sigma'\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(\mu x N)$. Finally by rule (GEN),

$$\langle UA'_x, \mathrm{Gen}(UA'_x, U\sigma')\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(\mu x N).$$

v) If $M = (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ x_{i_0})$ where $i_0 \in I$,
then $\langle A, \sigma\rangle = \langle A''', \mathrm{Gen}(A''', U\sigma_{i_0})\rangle$, where
$\langle A_i, \sigma_i\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M_i)$ for $i \in I$,
$A' = \sum_{i\in I} A_i$,
$A'' = A' \cup \{x_i : t_i \mid i \in I,\ x_i \notin \mathbf{dom}(A'),\ t_i \text{ fresh}\}$
$U \in \mathbf{MGS}(\{\sigma_i \le A''(x_i) \mid i \in I, \})$,
and $A''' = UA''_{\{x_i \mid i\in I\}}$,

By induction, $\langle A_i, \sigma_i\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M_i)$ for $i \in I$.

By weakening and substitutivity, $\langle UA'', U\sigma_i\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M_i)$ for $i \in I$.

Then by rule (LETREC-VAR), $\langle A''', U\sigma_{i_0}\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$, and by (GEN), $\langle A, \sigma\rangle = \langle A''', \mathrm{Gen}(A''', U\sigma_{i_0})\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$.

vi) If $M = (\textbf{letrec } x_i = M_i \mid i \in I \textbf{ in } N)$ where $N \notin \{x_i \mid i \in I\}$, the result follows by Lemma 67 and induction.

□

Theorem 70 (Principal pairs for $\mathbf{P}_2^{\mathrm{R}}$) *If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$, then there is a pair $\langle A', \sigma'\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$ and a substitution S such that $S\langle A', \sigma'\rangle \leq \langle A, \sigma\rangle$.*

Proof: By induction on the definition of $\mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$. For the rules of $\mathbf{P}_2$, see the proof of Theorem 59. We only need to consider the following cases.

vii) If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$ by rule (REC), then $M = (\mu x N)$, and for some $\tau \in \mathbf{T}_1$, we have $\langle A \cup \{x : \tau\}, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$ and $\sigma \leq_{\forall 2,1} \tau$.

By induction, we have a pair $\langle A', \sigma'\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(N)$ and a substitution S such that

$$S\langle A', \sigma'\rangle \leq \langle A \cup \{x : \tau\}, \sigma\rangle. \tag{3}$$

We consider two cases.

a) If $x \notin \mathbf{dom}(A')$, let t be fresh and $\pi = \{\sigma' \leq t\}$. Now $S\sigma' \leq_{\forall 2} \sigma \leq_{\forall 2,1} \tau$ by (3), and since $\tau \in \mathbf{T}_1$, there must be some $\tau' \in \mathbf{T}_0$ such that $\tau \leq_{\forall 2,1} \tau'$. Then $S\sigma' \leq_{\forall 2,1} \tau'$, so $S' = S \cup \{t := \tau'\}$ is a solution to π, and we may pick $U \in \mathbf{MGS}(\pi)$. Then

$$\langle UA', \mathrm{Gen}(UA', U\sigma')\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(\mu x N).$$

By Convention 35, there exists an R such that $RUA' = S'A' = SA$ and $RU\sigma' = S'\sigma' = S\sigma' \leq_{\forall 2} \sigma$. Furthermore by Lemma 47(vi) we have $R(\mathrm{Gen}(UA', U\sigma') \leq_{\forall 2} \sigma$, so that

$$R\langle UA', \mathrm{Gen}(UA', U\sigma')\rangle \leq \langle A, \sigma\rangle,$$

as desired.

b) If $x \in \mathbf{dom}(A')$, and $\tau' = A'(x)$, then by (3), $S\sigma' \leq_{\forall 2} \sigma \leq_{\forall 2,1} \tau \leq_1 S\tau'$, so S is a solution to $\pi = \{\sigma' \leq \tau'\}$.

Then pick $U \in \mathbf{MGS}(\pi)$, so that

$$\langle UA'_x, \mathrm{Gen}(UA'_x, U\sigma')\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(\mu x N).$$

By Convention 35, there exists an R such that $RUA'_x = SA'_x$ and $RU\sigma' = S\sigma'$. By (3), $A \leq_1 SA'_x$ and by (3) and Lemma 47(vi), $R(\mathrm{Gen}(UA', U\sigma')) \leq_{\forall 2} \sigma$. Therefore

$$R\langle UA'_x, \mathrm{Gen}(UA'_x, U\sigma')\rangle \leq \langle A, \sigma\rangle,$$

as desired.

viii) If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$ by rule (LETREC-VAR),

then $M = (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ x_{i_0})$ for some $i_0 \in I$, and for some $A_0 = \{x_i : \tau_i \mid i \in I\}$, we have $\sigma = \sigma_{i_0}$, and $\langle A \cup A_0, \sigma_i\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M_i)$ and $\sigma_i \leq_{\forall 2,1} \tau_i$ for all $i \in I$.

By induction, for all $i \in I$ we have disjoint pairs $\langle A_i, \sigma'_i\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M_i)$ and substitutions S_i such that

$$S_i\langle A_i, \sigma'_i\rangle \quad \leq \quad \langle A \cup A_0, \sigma_i\rangle. \tag{4}$$

Then $S_i\sigma'_i \leq_{\forall 2,1} \tau_i$, and since each $\tau_i \in \mathbf{T}_1$, there must be a $\tau'_i \in \mathbf{T}_0$ such that $\tau_i \leq_1 \tau'_i$, and thus $S_i\sigma'_i \leq_{\forall 2,1} \tau'_i$.

Let $A' = (\sum_{i \in I} A_i)$, $A'' = A' \cup \{x_i : t_i \mid i \in I,\ x_i \notin \mathbf{dom}(A'),\ t_i \text{ fresh}\}$, $\pi = \{\sigma'_i \leq A''(x_i) \mid i \in I\}$, and $S = \{t_i := \tau'_i \mid i \in I,\ x_i \notin \mathbf{dom}(A')\} \cup (\bigcup_{i \in I} S_i)$.

By (4), if $x_i \in \mathbf{dom}(A')$, then

$$S\sigma'_i \leq_{\forall 2} \sigma_i \leq_{\forall 2,1} \tau_i \leq_1 SA'(x_i) = SA''(x_i).$$

Otherwise $x_i \notin \mathbf{dom}(A')$ and

$$S\sigma'_i \leq_{\forall 2,1} \tau'_i = SA''(x_i).$$

Therefore S is a solution to π.

Pick $U \in \mathbf{MGS}(\pi)$ and let $A''' = UA'_{\{x_i \mid i \in I\}}$. Then

$$\langle A''', \mathrm{Gen}(A''', U\sigma'_{i_0})\rangle \in \mathrm{PP}_{\mathbf{P}_2^{\mathrm{R}}}(M).$$

By Convention 35, there exists an R such that $RU\sigma'_{i_0} = S\sigma'_{i_0} \leq_{\forall 2} \sigma_{i_0} = \sigma$, and

$$RA''' = RUA'_{\{x_i|i\in I\}} = SA'_{\{x_i|i\in I\}} \preceq_1 A.$$

By Lemma 47(vi), $R(\mathrm{Gen}(A''', U\sigma'_{i_0})) \leq_{\forall 2} \sigma$, so

$$\langle A''', \mathrm{Gen}(A''', U\sigma'_{i_0})\rangle \leq \langle A, \sigma\rangle.$$

ix) If $\langle A, \sigma\rangle \in \mathrm{AP}_{\mathbf{P}_2^{\mathrm{R}}}(M)$ by rule (LETREC), the result follows by Lemma 67 and induction.

□

6.6 Comparing (LETREC) and (LETREC-SIMPLE)

Now we show that our rules for typing recursive definitions are at least as powerful as the usual ones for ML.

We write $\mathrm{ML}^{\mathrm{R}} \rhd A \vdash M : \tau$ if $A \vdash M : \tau$ follows by the rules of ML and the rules (REC-SIMPLE) and (LETREC-SIMPLE), and $\Lambda_2^{\mathrm{R}} \rhd A \vdash M : \tau$ if $A \vdash M : \tau$ follows by the rules of Λ_2^{s} and (REC-SIMPLE) and (LETREC-SIMPLE).

It is well known (cf. [14]) that Λ_2^{R} types strictly more terms than ML^{R}.

Theorem 71 (Comparison of ML^{R} and Λ_2^{R}) *If $\mathrm{ML}^{\mathrm{R}} \rhd A \vdash M : \tau$, then $\Lambda_2^{\mathrm{R}} \rhd A \vdash M : \tau$. The converse does not hold.*

To show the relationship between Λ_2^{R} and $\mathbf{P}_2^{\mathrm{R}}$, we first state the following result, without proof.

Lemma 72 *If $M = (\mathbf{letrec}\ B\ \mathbf{in}\ N)$, then $\Lambda_2^{\mathrm{R}} \rhd A \vdash M : \sigma$ iff $\Lambda_2^{\mathrm{R}} \rhd A \vdash \langle\!\langle M \rangle\!\rangle : \sigma$.*

Theorem 73 (Comparison of Λ_2^{R} and $\mathbf{P}_2^{\mathrm{R}}$) *If $\Lambda_2^{\mathrm{R}} \rhd A \vdash M : \tau$, then $\mathbf{P}_2^{\mathrm{R}} \rhd A' \vdash M : \tau'$, where $A \preceq_1 A'$ and $\tau \preceq_2 \tau'$.*

Proof: We will use the following facts, which we state without proof:

- If $\tau \preceq_1 \tau'$ and $\tau \in \mathbf{T}_0$, then $\tau = \tau'$.
- If $\tau \preceq_2 \tau'$ and $\tau \in \mathbf{T}_0$, then $\tau = \tau'$.
- If $A \preceq_1 A'$, then $SA \preceq_1 SA'$.

- If $\Lambda_2^{\mathrm{R}} \triangleright A \vdash M : \sigma$ and $\mathrm{Gen}(A, \sigma) \succ \sigma'$, then $\Lambda_2^{\mathrm{R}} \triangleright A \vdash M : \sigma'$.

We prove the theorem by induction on M. By Lemmas 67 and 72, we need not consider the case $M = (\mathbf{letrec}\ B\ \mathbf{in}\ N)$ where N is not a variable defined in B. The cases $M = x$, $M = (\lambda x N)$, and $M = (M_1 M_2)$ can be proved just as in Theorem 12, and the case $M = (\mu x N)$ is trivial. That leaves only the following case.

- $M = (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ x_{i_0})$, where $i_0 \in I$.

  Then for some $\mathbf{T}_0$ type environment $A_0 = \{x_j : \tau_j \mid j \in I\}$, we have

  $$\Lambda_2^{\mathrm{R}} \triangleright A \cup A_0 \vdash M_i : \tau_i$$

  for all $i \in I$, and $\mathrm{Gen}(A, \tau_{i_0}) \succ \tau$.

  By induction, for all $i \in I$ we have

  $$\mathbf{P}_2^{\mathrm{R}} \triangleright A'_i \vdash M_i : \tau'_i,$$

  where $(A \cup A_0) \preceq_1 A'_i$ and $\tau_i \preceq_2 \tau'_i$. Since $\tau_i \in \mathbf{T}_0$ we have $\tau_i = \tau'_i$.

  Let $A' = (\sum_{i \in I} A'_i) + A_0$; then $(A \cup A_0) \preceq_1 A'$, and $A'(x_j) = \tau_j$ for any $j \in I$. By weakening, $\mathbf{P}_2^{\mathrm{R}} \triangleright A' \vdash M_i : \tau_i$ for all $i \in I$. By rule (LETREC-VAR),

  $$\mathbf{P}_2^{\mathrm{R}} \triangleright A'' \vdash (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ x_{i_0}) : \tau_{i_0},$$

  where $A'' = A'_{\{x_j \mid j \in I\}}$. Since $\mathrm{Gen}(A, \tau_{i_0}) \succ \tau$, for some substitution S we have $S\tau_{i_0} = \tau$ and $\mathbf{dom}(S) = \mathrm{FTV}(\tau_{i_0}) - \mathrm{FTV}(A)$. By substitutivity,

  $$\mathbf{P}_2^{\mathrm{R}} \triangleright SA'' \vdash (\mathbf{letrec}\ \{x_i = M_i \mid i \in I\}\ \mathbf{in}\ x_{i_0}) : \tau,$$

  and $A \preceq_1 A'' \Rightarrow A = SA \preceq_1 SA''$.

□

7 Compiling with rank 2 intersection types

We briefly discuss some applications of rank 2 intersections in compilation.

Polymorphism allows a function F of type $\forall t.t \rightarrow t$ to be applied to arguments of any type. Unfortunately, it also requires that the data representation of its arguments be reduced to a lowest common denominator: the machine code for F cannot handle both a 32-bit integer in a general purpose register and a 64-bit floating point number in a float register. In practice, arguments are "boxed," or represented as a pointer to the actual data value stored in main memory. Boxing and unboxing coercions slow program execution.

These overheads can be reduced when more is known about the uses of the polymorphic function. For example, consider the program

$$M = (\lambda f.(f\,3, f\,\mathbf{true}))F.$$

A naive implementation would insert instructions to box the arguments 3 and **true** before passing them to F. A more clever implementation would recognize that the only arguments of F are integers and booleans, both of which can be represented in a single 32-bit register; so F could be compiled to expect an unboxed value as its argument.

This can easily be achieved in $\mathbf{P}_2$. To compile M, we first calculate the principal typings of the operator and operand:

$$\begin{array}{rcl} (\lambda f.(f\,3, f\,\mathbf{true})) & : & \forall s,u.(\mathrm{int} \to s) \wedge (\mathrm{bool} \to u) \to s \times u, \\ F & : & \forall t.t \to t. \end{array}$$

The type of the operator indicates that F will only be applied to integers and booleans, and the compiler can take advantage of this in generating the machine code for F. Note that this improves on Bjørner's *minimal typing derivations* [3], which would require the arguments to be boxed.

$\mathbf{P}_2$ also supports other data representation strategies. For example, in compiling the program $(\lambda f.(f\ 3, f\ 2.4))F$, we will calculate the principal typing

$$(\lambda f.(f\,3, f\,2.4)) \quad : \quad \forall s,u.(\mathrm{int} \to s) \wedge (\mathrm{float} \to u) \to s \times u.$$

If floating point numbers are 64-bit values, we can't just compile F to expect its argument in a 32-bit register, as before. Boxing is one solution. But another solution is possible: *specialization* [8]. We can generate *two* versions of F, one expecting an unboxed integer in a 32-bit register, and one expecting an unboxed float in a 64-bit register. We are essentially *overloading* the variable f, so the application $(f\ 3)$ invokes the integer-expecting F, and $(f\,2.4)$ invokes the float-expecting F.

8 Conclusion

We discussed a variety of rank 2 type systems: Λ_2, the rank 2 fragment of System F; $\mathbf{I}_2$, $\mathbf{I}_2^{\mathrm{s}}$, and $\mathbf{I}_2^{\mathrm{yb}}$, all variants of the rank 2 intersection type discipline; and $\mathbf{P}_2$, which adds ML-style, top-level quantification of type variables to $\mathbf{I}_2$. We showed that all of the systems are equivalent in terms of typability—a term is typable in one system if and only if it is typable in another. An immediate corollary is that typability in all of these systems is DEXPTIME-complete. We have also determined that the sequence $\mathbf{I}_2^{\mathrm{vb}}$, $\mathbf{I}_2^{\mathrm{s}}$, $\mathbf{I}_2$, $\mathbf{P}_2$ is in order of increasing "expressiveness." For example, a judgment of $\mathbf{I}_2^{\mathrm{vb}}$ is a judgment of $\mathbf{P}_2$, but not vice versa.

We proposed a new rule for typing recursive definitions that can type many examples of polymorphic recursion. The extension of $\mathbf{P}_2$ by this rule results in a system with principal typings and decidable type inference.

Finally, we discussed some applications of intersections in compilation. The finite polymorphism of intersections expresses data representation constraints more accurately than polymorphism by quantification. The accurate expression of these constraints leads to data representations that require fewer boxing and unboxing coercions at runtime.

Acknowledgments. Albert Meyer, Jens Palsberg, and Mona Singh made many useful suggestions. I would particularly like to thank Assaf Kfoury for his careful reading of a draft.

References

[1] Franz Baader and Jörg Siekmann. Unification theory. In Dov M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2, pages 41–125. Claredon Press, 1994.

[2] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics (Revised Edition)*, volume 103 of *Studies in Logic and the Foundation of Mathematics*. North-Holland, 1984.

[3] Nikolaj Skallerud Bjørner. Minimal typing derivations. In *Record of the 1994 ACM SIGPLAN Workshop on ML and its Applications*, pages 120–126, June 1994.

[4] Dominique Clement, Joelle Despeyroux, Thierry Despeyroux, and Gilles Kahn. A simple applicative language: Mini-ML. In *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, pages 13–27, 1986.

[5] M. Coppo and M. Dezani. A new type-assignment for lambda terms. *Archiv für Math. Logik*, 19:139–156, 1978.

[6] J.H. Gallier and W. Snyder. Designing unification procedures using transformations: A survey. In Y.N. Moschovakis, editor, *Logic from Computer Science*, volume 21 of *Mathematical Sciences Research Institute Publications*, pages 153–215. Springer-Verlag, 1992.

[7] Jean-Yves Girard. Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J.E. Fenstad, editor, 2^{nd} *Scandinavian Logic Symp.*, pages 63–92. North-Holland Publishing Co., 1971.

[8] Cordelia V. Hall, Simon L. Peyton Jones, and Patrick M. Sansom. Unboxing using specialisation. In *Glasgow Functional Programming Workshop*. Springer-Verlag, July 1994.

[9] Fritz Henglein. Type inference with polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):253–289, April 1993.

[10] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez

and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, chapter 8, pages 257–321. MIT Press, 1991.

[11] A.J. Kfoury and J. Tiuryn. Type reconstruction in finite-rank fragments of the polymorphic λ-calculus (extended summary). In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 2–11. IEEE Computer Society Press, 1990.

[12] A.J. Kfoury and J. Tiuryn. Type reconstruction in finite rank fragments of the second-order λ-calculus. *Information and Computation*, 98(2):228–257, June 1992.

[13] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. A proper extension of ML with an effective type-assignment. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 58–69, 1988.

[14] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. Type reconstruction in the presence of polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):290–311, April 1993.

[15] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *Journal of the ACM*, 41(2), March 1994.

[16] A.J. Kfoury and J.B. Wells. A direct algorithm for type inference in the rank 2 fragment of the second-order lambda-calculus. Technical Report 93–017, Boston University, November 1993.

[17] A.J. Kfoury and J.B. Wells. A direct algorithm for type inference in the rank 2 fragment of the second-order lambda-calculus. In *Proceedings of the 1994 ACM Conference on Lisp and Functional Programming*, pages 196–207, 1994.

[18] A.J. Kfoury and J.B. Wells. New notions of reduction and non-semantic proofs of strong β-normalization in typed λ-calculi. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 311–321. IEEE Computer Society Press, 1995.

[19] Kevin Knight. Unification: A multidisciplinary survey. *ACM Computing Reviews*, 21(1):93–124, March 1989.

[20] J.-L. Lassez, M.J. Maher, and K. Marriot. Unification revisited. In Jack Minker, editor, *Deductive Databases and Logic Programming*, chapter 15, pages 587–625. Morgan Kaufman, 1988.

[21] Daniel Leivant. Polymorphic type inference. In *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages*, pages 88–98, 1983.

[22] Harry G. Mairson. Deciding ML typability is complete for deterministic exponential time. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 382–401, 1990.

[23] Nancy McCracken. The typechecking of programs with implicit type structure. In G. Kahn, D.B. MacQueen, and G. Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 301–315, June 1984.

[24] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, 1990.

[25] A. Mycroft. Polymorphic type schemes and recursive definitions. In *Proceedings of the International Symposium on Programming, Toulouse*, volume 167 of *Lecture Notes in Computer Science*, pages 217–239. Springer-Verlag, 1984.

[26] Gordon D. Plotkin. A note on inductive generalization. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 5*, 1970.

[27] John C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 5*, pages 135–151, 1970.

[28] John C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium: Proceedings, Colloque sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag, 1974.

[29] Patrick Sallé. Une extension de la theorie des types en λ-calcul. In G. Ausiello and C. Böhm, editors, *Automata, Languages and Programming: Fifth Colloquium*, volume 62 of *Lecture Notes in Computer Science*, pages 398–410. Springer-Verlag, July 1978.

[30] Jörg H. Siekmann. Unification theory. *J. Symbolic Computation*, 7(3&4):207–274, March/April 1989.

[31] Wayne Snyder. *A Proof Theory for General Unification*, volume 11 of *Progress in Computer Science and Applied Logic*. Birkhäuser, 1991.

[32] Mads Tofte. *Operational Semantics and Polymorphic Type Inference.* PhD thesis, University of Edinburgh, May 1988.

[33] Steffen van Bakel. *Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems.* PhD thesis, Mathematisch Centrum, Amsterdam, February 1993.

[34] J.B. Wells. Typability and type checking in the second-order λ-calculus are equivalent and undecidable. Technical Report 93–011, Boston University, 1993.

[35] Hirofumi Yokouchi. Embedding a second-order type system into an intersection type system. *Information and Computation*, 117(2):206–220, March 1995.

Index