System F with Width-subtyping and Record Updating *

Erik Poll

Computing Laboratory, University of Kent, Canterbury, England

Abstract. It is a well-known problem that $F_{\leq}$ – the polymorphic lambda calculus F extended with subtyping – does not provide so-called *polymorphic updates*, and that the standard PER model for $F_{\leq}$ does not provide interpretations for these operations. The polymorphic updates are interesting because they play an important role in some type-theoretic models of object-oriented languages. We present an extension F^{width} of system F with a restricted form of subtyping – *width-subtyping* – on record types, that does provide these operations. The main result is that we show it is still possible to give a PER model for this system.

1 Introduction

There have been many attempts to model object-oriented languages in typed lambda calculi (see for instance [CW85], many of the papers in [GM94], [FM94], or [AC96]). The type systems used for these object models are usually variants of $F_{\leq}$, the extension of the polymorphic lambda calculus – system F – with subtyping introduced in [CW85]. Unfortunately, $F_{\leq}$ has the well-known deficiency that it does not provide so-called *polymorphic record-updates*, discussed in more detail below. These operations play an important role in some object models, notably in the existential object model introduced in [PT94].

One solution to this problem has been the introduction of richer systems for record types and operations on records, e.g. [CM91] [Car92] [Zwa95]. But these systems are very expressive and (hence) rather complicated.

Another approach is taken in [HP96], where subtyping is restricted to so-called *positive* subtyping. We go one step further and restrict this notion of positive subtyping to *width-subtyping* on record types, resulting in a system F^{width}. The intended application of F^{width} – like that of [HP96] – is the existential object model of [PT94]. Width-subtyping has several advantages over positive subtyping, notably the much simpler operational semantics and denotational PER semantics.

The syntax of F^{width} is given in Section 2. The main challenge is to provide a semantics for F^{width}, because the standard (PER) model construction for system F seems to rule out polymorphic record-updates. However, we show that it is

* to appear in: *Proceedings of Theoretical Aspects of Computer Software (TACS'97), Sendai , Japan*

possible to extend the standard PER model to interpret F^{width} in Section 3. Section 4 gives a comparison with related work. We point out some possible extensions of the system in Section 5 and conclude in Section 6. The rest of this section discusses polymorphic record-updates, their relevance for modelling objects, why they cannot be typed using subtyping, and how they can be typed using width-subtyping.

Typing Polymorphic Record Updates

Polymorphic Record Updates. Because we are in a functional setting, updating a record means making a copy of a record with one or more of its fields changed. An example of a function that updates a record is a function `have_birthday` that takes a record of type $\langle \texttt{age}:\texttt{Nat}, \texttt{name}:\texttt{String}\rangle$ as input and returns the record with its `age`-field increased by 1. Similar functions exist of course for all record types that include such an `age`-field, and we would like to be able to write a single *generic* or *polymorphic* function `have_birthday` that can be applied to *any* record with an `age`-field of type `Nat`. This requires a record-update that can be applied to records of many different types – viz. all record types with an `age`-field of type `Nat` – which is known as a *polymorphic* record-update.

Polymorphic Updates and Objects. To understand the use for polymorphic updates for modelling objects, suppose that an object is modelled as a piece of state – a record of instance-variables – together with a collection of functions – the methods – that act on this state. For example, objects of a class `AGE` could have states of type $\langle \texttt{age}:\texttt{Nat}\rangle$ and `have_birthday` as one of their methods. Objects in subclasses will have have richer states, i.e. states with more instance variables. For example, objects of a subclass `PERSON` of `AGE` could have states of type $\langle \texttt{name}:\texttt{String}, \texttt{age}:\texttt{Nat}\rangle$. A method of a superclass should be applicable to these richer states of objects in a subclass, e.g. `have_birthday` should be applicable to the states of `PERSON`'s. This means we want the polymorphic `have_birthday` discussed above as method of `AGE`.

The Problem with Subtyping. The subtyping relation of $F_{\leq}$ captures the notion of *substitutivity*: a type σ is a subtype of τ – written $\sigma \leq \tau$ – if an expression of type σ can be used whenever an expression of type τ is required, without introducing type errors. Unfortunately, this notion of subtyping turns out to be too weak to type polymorphic updates such as the `have_birthday` above. At first sight one expects that a good type for `have_birthday` would be $\forall\sigma \leq \langle \texttt{age}:\texttt{Nat}\rangle.\, \sigma \to \sigma$. But this is *not* the case! The problem is that there may be subtypes σ of $\langle \texttt{age}:\texttt{Nat}\rangle$ – for example $\langle \texttt{age}:\texttt{Even}\rangle$ – for which increasing the `age`-field of a term of type σ by 1 does not produce a result of type σ. The standard PER model of $F_{\leq}$ does provide all subsets of $\mathbb{N}$ as subtypes of `Nat`, which means that in this model $\forall\sigma \leq \langle \texttt{age}:\texttt{Nat}\rangle.\, \sigma \to \sigma$ has an identity function as its only element (see [BL90]).

Width-Subtyping. Basically, the problem is that there are too many subtypes.

Subtyping includes not only so-called *width*-subtyping

$$\frac{m \geq n}{\langle l_1{:}\sigma_1, \ldots, l_m{:}\sigma_m\rangle \leq \langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle}\ (\textsc{width})$$

but also *depth*-subtyping

$$\frac{\sigma_i \leq \tau_i \text{ for all } i = 1 \ldots n}{\langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle \leq \langle l_1{:}\tau_1, \ldots, l_n{:}\tau_n\rangle}\ (\textsc{depth})$$

We will solve the problem of typing `have_birthday` by considering width-subtyping in isolation. A type σ is a width-subtype of τ – written $\sigma \sqsubseteq \tau$ – if σ can be obtained from τ by adding fields. We write $M\langle l := N\rangle$ for the record M with its l-field changed to N and all its other fields unchanged. The typing rule for this update operation is

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau \quad \sigma \sqsubseteq \langle l{:}\tau\rangle}{\Gamma \vdash M\langle l := N\rangle : \sigma}\ (\textsc{update})$$

So, for example, $\lambda x{:}\sigma.\, x\langle \mathsf{age} := x.\mathsf{age} + 1\rangle : \sigma \to \sigma$ for any $\sigma \sqsubseteq \langle \mathsf{age}{:}\mathtt{Nat}\rangle$, and the polymorphic `have_birthday` is obtained by abstracting over all $\sigma \sqsubseteq \langle \mathsf{age}{:}\mathtt{Nat}\rangle$:

$$\begin{aligned}\mathtt{have\_birthday} &= \lambda\sigma \sqsubseteq \langle \mathsf{age}{:}\mathtt{Nat}\rangle.\, \lambda x{:}\sigma.\, x\langle \mathsf{age} := x.\mathsf{age} + 1\rangle \\ &: \forall \sigma \sqsubseteq \langle \mathsf{age}{:}\mathtt{Nat}\rangle.\, \sigma \to \sigma \quad .\end{aligned}$$

Polymorphic functions being *parametric* – which roughly means they behave in the same way at different types – we expect this type $\forall \sigma \sqsubseteq \langle \mathsf{age}{:}\mathtt{Nat}\rangle.\, \sigma \to \sigma$ to be isomorphic to $\mathtt{Nat} \to \mathtt{Nat}$. This isomorphism does indeed hold in the PER model given in Sect. 3 (Lemma 34).

2 The System F^{width}

We add 4 new term constructions to system F: records $\langle l_1 = M_1, \ldots, l_n = M_n\rangle$, field-selections $M.l$, record-updates $M\langle l := N\rangle$, and width-bounded abstractions $(\lambda\alpha \sqsubseteq \sigma.\, M)$. We add 2 new type constructions: record types $\langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle$ and width-bounded quantifications $(\forall\alpha \sqsubseteq \sigma.\, \tau)$.

Definition 1. The *terms* M and *types* σ of F^{width} are given by the grammar

$$\begin{aligned} M &::= x \mid \lambda x{:}\sigma.\, M \mid MM \mid \lambda\alpha.\, M \mid M\sigma \\ &\quad\ \mid \langle l = M, \ldots, l = M\rangle \mid M.l \mid M\langle l := M\rangle \mid \lambda\alpha \sqsubseteq \sigma.\, M \\ \sigma &::= \alpha \mid \sigma \to \sigma \mid \forall\alpha.\, \sigma \mid \langle l{:}\sigma, \ldots, l{:}\sigma\rangle \mid \forall\alpha \sqsubseteq \sigma.\, \sigma \end{aligned}$$

Here x ranges over *term-variables*, α over *type-variables*, and l over a countable set of *labels*. Free and bound variables are defined as usual. Terms and types equal up to the names of bound variables and permutation of fields are identified. We assume that in $\langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle$ and $\langle l_1 = M_1, \ldots, l_n = M_n\rangle$ no label l_i occurs twice. We write $[e/x]e'$ for the capture-free substitution of e for x in e'.

The *contexts* of F^{width} are given by

$$\Gamma ::= \epsilon \mid \Gamma, x : \sigma \mid \Gamma, \alpha : \mathit{Type} \mid \Gamma, \alpha \sqsubseteq \sigma$$

with the restriction that no variable may be declared twice, and that in $\Gamma, x{:}\sigma$ and $\Gamma, \alpha \sqsubseteq \sigma$ all free type variables in σ must be declared in Γ.

We write $\Gamma \vdash \sigma : \mathit{Type}$ if all free type variables in σ are declared in Γ.

Definition 2. The *width-subtype relation* $\Gamma \vdash \sigma \sqsubseteq \tau$ is the smallest relation closed under the following rules:

$$\frac{}{\Gamma, \alpha \sqsubseteq \tau, \Gamma' \vdash \alpha \sqsubseteq \tau}\ (\sqsubseteq\text{-CONTEXT})$$

$$\frac{m \geq n \quad \Gamma \vdash \sigma_i : \mathit{Type} \text{ for all } i = 1 \ldots m}{\Gamma \vdash \langle l_1{:}\sigma_1, \ldots, l_m{:}\sigma_m\rangle \sqsubseteq \langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle}\ (\sqsubseteq\text{-WIDTH})$$

$$\frac{\Gamma \vdash \rho \sqsubseteq \langle\rangle}{\Gamma \vdash \rho \sqsubseteq \rho}\ (\sqsubseteq\text{-REFL}) \qquad \frac{\Gamma \vdash \rho \sqsubseteq \sigma \quad \Gamma \vdash \sigma \sqsubseteq \tau}{\Gamma \vdash \rho \sqsubseteq \tau}\ (\sqsubseteq\text{-TRANS})$$

Note that width-subtyping is only defined on record types. $\Gamma \vdash \rho \sqsubseteq \langle\rangle$ means that ρ is a record type, so the rule ($\sqsubseteq$-REFL) states that $\sqsubseteq$ is only reflexive on the record types.

Definition 3. The *typing relation* $\Gamma \vdash M : \sigma$ of F^{width} is the smallest relation closed under the type inference rules of F

$$\frac{}{\Gamma, x : \sigma, \Gamma' \vdash x : \sigma}\ (\text{VAR})$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x{:}\sigma.\, M : \sigma \to \tau}\ (\to\text{-INTRO}) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\ (\to\text{-ELIM})$$

$$\frac{\Gamma, \alpha{:}\mathit{Type} \vdash M : \sigma}{\Gamma \vdash \lambda\alpha.\, M : \forall\alpha.\, \sigma}\ (\forall\text{-INTRO}) \qquad \frac{\Gamma \vdash M : \forall\alpha.\, \sigma \quad \Gamma \vdash \tau : \mathit{Type}}{\Gamma \vdash M\tau : [\tau/\alpha]\sigma}\ (\forall\text{-ELIM})$$

plus the additional rules

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash \sigma \sqsubseteq \tau}{\Gamma \vdash M : \tau}\ (\sqsubseteq\text{-SUB})$$

$$\frac{\Gamma \vdash M_1 : \sigma_1 \quad \ldots \quad \Gamma \vdash M_n : \sigma_n}{\Gamma \vdash \langle l_1 = M_1, \ldots, l_n = M_n\rangle : \langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle}\ (\text{RECORD-INTRO})$$

$$\frac{\Gamma \vdash M : \langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle \quad l_i \in \{l_1, \ldots, l_n\}}{\Gamma \vdash M.l_i : \sigma_i}\ (\text{RECORD-ELIM})$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau \quad \Gamma \vdash \sigma \sqsubseteq \langle l{:}\tau\rangle}{\Gamma \vdash M\langle l := N\rangle : \sigma} \quad (\textsc{update})$$

$$\frac{\Gamma, \alpha \sqsubseteq \rho \vdash M : \sigma}{\Gamma \vdash \lambda\alpha \sqsubseteq \rho.\, M : \forall\alpha \sqsubseteq \rho.\, \sigma} \quad (\forall_{\sqsubseteq}\textsc{-intro})$$

$$\frac{\Gamma \vdash M : \forall\alpha \sqsubseteq \rho.\, \sigma \quad \Gamma \vdash \tau \sqsubseteq \rho}{\Gamma \vdash M\tau : [\tau/\alpha]\sigma} \quad (\forall_{\sqsubseteq}\textsc{-elim})$$

Definition 4. The *β-reduction relation* $\rhd_\beta$ on terms is given by the rewrite rules

$$\begin{aligned}
(\lambda x{:}\sigma.\, M)N &\rhd_\beta [N/x]M \\
(\lambda\alpha.\, M)\tau &\rhd_\beta [\tau/\alpha]M \\
(\lambda\alpha \sqsubseteq \rho.\, M)\tau &\rhd_\beta [\tau/\alpha]M \\
\langle l_1 = M_1, \ldots, l_n = M_n\rangle.l_i &\rhd_\beta M_i \\
M\langle l := N\rangle.l' &\rhd_\beta \begin{cases} M.l' & \text{if } l \neq l' \\ N & \text{if } l = l' \end{cases}
\end{aligned}$$

We write $\rhd\!\!\!\rhd_\beta$ for the reflexive and transitive closure of $\rhd_\beta$, and $\Gamma \vdash M \rhd_\beta N : \sigma$ as abbreviation for $M \rhd_\beta N$ and $\Gamma \vdash M : \sigma$ and $\Gamma \vdash N : \sigma$.

Theorem 5 (Church-Rosser). *If $M \rhd_\beta M_1$ and $M \rhd_\beta M_2$ then $M_1 \rhd\!\!\!\rhd_\beta N$ and $M_2 \rhd\!\!\!\rhd_\beta N$ for some term N.*

Proof. Standard. □

Lemma 6 (Generation). *1. If $\Gamma \vdash \langle l_1 = M_1, \ldots, l_n = M_n\rangle : \sigma$, then $\sigma = \langle l'_1{:}\sigma_1, \ldots, l'_m{:}\sigma_m\rangle$ with $\{l_1, \ldots, l_n\} \supseteq \{l'_1, \ldots, l'_m\}$ and $\Gamma \vdash M_i : \sigma_j$ for all $l_i = l'_j \in \{l'_1, \ldots, l'_m\}$.*
2. If $\Gamma \vdash M\langle l := N\rangle : \sigma$, then $\Gamma \vdash M : \sigma$, $\sigma = \langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle$, and $\Gamma \vdash N : \sigma_i$ if $l = l_i \in \{l_1, \ldots, l_n\}$.

Proof. Induction on the derivation (which can only end with ($\sqsubseteq$-SUB) or with (RECORD-INTRO) for 1, and with ($\sqsubseteq$-SUB) or (UPDATE) for 2.). □

Lemma 7 (Substitution). *Let J be a judgement of the form $M{:}\sigma$ or $\sigma \sqsubseteq \tau$. Then*

1. *If $\Gamma, x{:}\rho, \Gamma' \vdash J$ and $\Gamma \vdash N : \rho$ then $\Gamma, [N/x]\Gamma' \vdash [N/x]J$.*
2. *If $\Gamma, \alpha{:}Type, \Gamma' \vdash J$ and $\Gamma \vdash \rho : Type$ then $\Gamma, [\rho/\alpha]\Gamma' \vdash [\rho/\alpha]J$.*
3. *If $\Gamma, \alpha \sqsubseteq \tau, \Gamma' \vdash M : \sigma$ and $\Gamma \vdash \rho \sqsubseteq \tau$ then $\Gamma, [\rho/\alpha]\Gamma' \vdash [\rho/\alpha]J$.*

Proof. Induction on the derivation of J. □

Theorem 8 (Subject Reduction). *If $\Gamma \vdash M : \sigma$ and $M \rhd_\beta N$ then $\Gamma \vdash N : \sigma$.*

Proof. By induction on the derivation of $\Gamma \vdash M : \sigma$ we simultaneously prove

1. $\Gamma \vdash M : \sigma \ \wedge \ M \rhd_\beta N \ \Rightarrow \ \Gamma \vdash N : \sigma$,
2. $\Gamma \vdash M : \sigma \ \wedge \ \Gamma \rhd_\beta \Gamma' \ \Rightarrow \ \Gamma' \vdash M : \sigma$.

The interesting cases are the cases of 1 where M is the redex. We treat one case in detail.

- Suppose the last step in the derivation is (RECORD-ELIM). If $M \rhd_\beta N$ is a reduction $M'.l \rhd_\beta N'.l$ with $M' \rhd_\beta N'$, then the proof is easy. If M itself is the redex, then there are two possibilities:
  - $M \rhd_\beta N$ is the reduction $\langle l_1 = M_1, \dots, l_n = M_n \rangle.l_i \rhd_\beta M_i$. Then the derivation ends with

    $$\frac{\Gamma \vdash \langle l_1 = M_1, \dots, l_n = M_n \rangle : \langle l_1{:}\sigma_1, \dots, l_m{:}\sigma_m \rangle^{(\mathrm{i})}}{\Gamma \vdash \langle l_1 = M_1, \dots, l_n = M_n \rangle.l_i : \sigma_i} \ (\textsc{record-elim})$$

    and by Lemma 6.1 it follows from (i) that $\Gamma \vdash M_i : \sigma_i$.
  - $M \rhd_\beta N$ is the reduction $M_1\langle l := M_2 \rangle.l_i \rhd_\beta \begin{cases} M_1.l_i & \text{if } l \neq l_i \\ M_2 & \text{if } l = l_i \end{cases}$

    This case is proved in roughly the same way, now using Lemma 6.2.

For the other β-reduction rules the substitution lemma (Lemma 7) is needed. $\square$

For records the notion of η-equality is type-dependent. E.g., M is η-equal to $\langle l = M.l \rangle$ if $M : \langle l{:}\sigma \rangle$, but not if $M : \langle l{:}\sigma, l'{:}\sigma' \rangle$. So we can only talk of $\beta\eta$-equality of *well-typed* terms *at a certain type*, which is written $\Gamma \vdash M =_{\beta\eta} N : \sigma$.

Definition 9. We define *$\beta\eta$-equality* (at a given type, in a given context) as the smallest equivalence relation – i.e. reflexive, symmetric, and transitive relation – closed under the β- and η-rules

$$\frac{\Gamma \vdash M : \sigma \quad M \rhd_\beta N}{\Gamma \vdash M =_{\beta\eta} N : \sigma}$$

$$\frac{\Gamma \vdash (\lambda x{:}\rho.\, M)x : \sigma \quad x \text{ not free in } M}{\Gamma \vdash (\lambda x{:}\rho.\, M)x =_{\beta\eta} M : \sigma}$$

$$\frac{\Gamma \vdash (\lambda \alpha.\, M)\alpha : \sigma \quad \alpha \text{ not free in } M}{\Gamma \vdash (\lambda \alpha.\, M)\alpha =_{\beta\eta} M : \sigma}$$

$$\frac{\Gamma \vdash (\lambda \alpha \sqsubseteq \rho.\, M)\alpha : \sigma \quad \alpha \text{ not free in } M}{\Gamma \vdash (\lambda \alpha \sqsubseteq \rho.\, M)\alpha =_{\beta\eta} M : \sigma}$$

$$\frac{\Gamma \vdash M : \langle l_1{:}\sigma_1, \dots, l_n{:}\sigma_n \rangle}{\Gamma \vdash \langle l_1 = M.l_1, \dots, l_n = M.l_n \rangle =_{\beta\eta} M : \langle l_1{:}\sigma_1, \dots, l_n{:}\sigma_n \rangle}$$

$$\frac{\Gamma \vdash M\langle l := M.l \rangle : \sigma}{\Gamma \vdash M\langle l := M.l \rangle =_{\beta\eta} M : \sigma}$$

the following congruence rule for subsumption

$$\frac{\Gamma \vdash M =_{\beta\eta} N : \sigma \quad \Gamma \vdash \sigma \sqsubseteq \tau}{\Gamma \vdash M =_{\beta\eta} N : \tau}$$

and finally congruence rules for each term constructor:

$$\frac{\Gamma, x : \sigma \vdash M =_{\beta\eta} M' : \tau}{\Gamma \vdash \lambda x{:}\sigma.\, M =_{\beta\eta} \lambda x{:}\sigma.\, M' : \sigma \to \tau} \qquad \frac{\Gamma \vdash M =_{\beta\eta} M' : \sigma \to \tau \quad \Gamma \vdash N =_{\beta\eta} N' : \sigma}{\Gamma \vdash MN =_{\beta\eta} M'N' : \tau}$$

$$\frac{\Gamma, \alpha{:}\mathit{Type} \vdash M =_{\beta\eta} M' : \tau}{\Gamma \vdash \lambda\alpha.\, M =_{\beta\eta} \lambda\alpha.\, M' : \forall\alpha.\,\tau} \qquad \frac{\Gamma \vdash M =_{\beta\eta} M' : \forall\alpha.\,\tau \quad \Gamma \vdash \rho : \mathit{Type}}{\Gamma \vdash M\rho =_{\beta\eta} M'\rho : [\rho/\alpha]\tau}$$

$$\frac{\Gamma, \alpha \sqsubseteq \sigma \vdash M =_{\beta\eta} M' : \tau}{\Gamma \vdash \lambda\alpha{\sqsubseteq}\sigma.\, M =_{\beta\eta} \lambda\alpha{\sqsubseteq}\sigma.\, M' : \forall\alpha{\sqsubseteq}\sigma.\,\tau} \qquad \frac{\Gamma \vdash M =_{\beta\eta} M' : \forall\alpha{\sqsubseteq}\sigma.\,\tau \quad \Gamma \vdash \rho \sqsubseteq \sigma}{\Gamma \vdash M\rho =_{\beta\eta} M'\rho : [\rho/\alpha]\tau}$$

$$\frac{\Gamma \vdash M =_{\beta\eta} M' : \langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle}{\Gamma \vdash M.l_i =_{\beta\eta} M'.l_i : \sigma_i}$$

$$\frac{\Gamma \vdash M_1 =_{\beta\eta} M_1' : \sigma_1 \quad \ldots \quad \Gamma \vdash M_n =_{\beta\eta} M_n' : \sigma_n}{\Gamma \vdash \langle l_1 = M_1, \ldots, l_n = M_n\rangle =_{\beta\eta} \langle l_1 = M_1', \ldots, l_n = M_n'\rangle : \langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle}$$

$$\frac{\Gamma \vdash M =_{\beta\eta} M' : \sigma \quad \Gamma \vdash N =_{\beta\eta} N' : \tau \quad \Gamma \vdash \sigma \sqsubseteq \langle l{:}\tau\rangle}{\Gamma \vdash M\langle l{:=}N\rangle =_{\beta\eta} M'\langle l{:=}N'\rangle : \sigma}$$

$$\frac{\Gamma \vdash M =_{\beta\eta} M' : \sigma \quad l_1 \neq l_2 \quad \Gamma \vdash N_i =_{\beta\eta} N_i' : \tau_i \text{ and } \Gamma \vdash \sigma \sqsubseteq \langle l_i{:}\tau_i\rangle \text{ for } i = 1, 2}{\Gamma \vdash M\langle l_1{:=}N_1\rangle\langle l_2{:=}N_2\rangle =_{\beta\eta} M'\langle l_2{:=}N_2'\rangle\langle l_1{:=}N_1'\rangle : \sigma}$$

Lemma 10. *If $\Gamma \vdash M = N : \sigma$ then $\Gamma \vdash M : \sigma$ and $\Gamma \vdash N : \sigma$.*

Proof. Induction on the derivation of $\Gamma \vdash M =_{\beta\eta} N : \sigma$, using the subject reduction property to deal with the β-rules. □

2.1 Application to the Existential Object Model

In the existential object model of [PT94] classes are polymorphic records of "pre-methods" that can be used either to create objects or to build sub-classes. These classes can be written in F^{width} exactly as in [HP96]. All the examples of class definitions given in [HP96] are immediately typable in F^{width}, so we will just give one of these and refer to [HP96] and [PT94] for more explanation. For example, a simple class of points with interface

$$M(\alpha) = \langle \mathtt{get} : \alpha \to \mathtt{Int}, \mathtt{set} : \alpha \to \mathtt{Int} \to \alpha, \mathtt{bump} : \alpha \to \alpha\rangle$$

and representation type $R = \langle x : \mathtt{Int}\rangle$ is given by

$$\begin{aligned}
\mathtt{PointClass} = \lambda\alpha \sqsubseteq \langle x : \mathtt{Int}\rangle.\, \lambda\mathtt{self} : M(\alpha).\\
&\langle \mathtt{get} = \lambda s : \alpha.\, s.x\\
&,\mathtt{set} = \lambda s : \alpha, i : \mathtt{Int}.\, s\langle x{:=}i\rangle\\
&,\mathtt{bump} = \lambda s : \alpha.\, \mathtt{self.set}\ s((\mathtt{self.get}\ s) + 1)\rangle\\
: \forall\alpha \sqsubseteq R.\, M(\alpha) \to M(\alpha).
\end{aligned}$$

Another use of width-subtyping is to model objects with public instance variables. In the existential object model the type of objects with interface M is $\mathtt{Object}(M) = \exists\alpha.\ \langle \mathtt{state} : \alpha, \mathtt{methods} : M(\alpha)\rangle$. Using a width-bounded quantification in this type we can expose some of the representation and make one or more instance variables public. For example, $\exists\alpha \sqsubseteq \langle x{:}\mathtt{Int}\rangle.\ \langle \mathtt{state} : \alpha, \mathtt{methods} : M(\alpha)\rangle$ is the type of objects with interface M that have a public instance variable x of type $\mathtt{Int}$.

3 PER Semantics

The PER model for F^{width} given below extends the standard PER model for system F. Types are interpreted as partial equivalence relation (pers) on $\mathbb{N}$, and terms as (indices of) partial recursive functions. If the per R is the interpretation of type σ, then interpretations of terms of type σ are equal if they are related by R.

The difficulty in modelling F^{width} is finding a suitable relation on pers to interpret width-subtyping. Width-subtyping is a "structural" subtype relation: the width-subtype of a record type is a record type. On the other hand, the interpretation of subtyping in the PER model for $F_{\leq}$ [BL90] is the "unstructured" subset relation on pers, which – as explained in the introduction – is precisely why it does not provide polymorphic updates. The interpretation of width-subtyping in the PER model is made possible by the fact that we can tell which pers are interpretations of record types.

Definition 11. A *partial equivalence relation* (*per*) is a relation that is symmetric and transitive. PER is the collection of partial equivalence relations over $\mathbb{N}$. We write dom_R for $\{n \in \mathbb{N} \mid (n, n) \in R\}$ and $\emptyset$ for the empty relation.

Definition 12. We assume some enumeration of the partial recursive functions, and write $n \cdot m$ for the application of the n^{th} partial recursive function to m. Application associates to the left. We write $n \cdot m \uparrow$ for "$n \cdot m$ is undefined", and $n \cdot m \downarrow$ for "$n \cdot m$ is defined". Whenever we write $(E, E') \in R$ or $E \in dom_R$ for certain expressions E and E', it is implicit that these expressions are defined.

3.1 The Interpretation of Terms

The interpretation of terms is a simple extension of the interpretation of terms in the standard PER model. Records are interpreted as in [BL90], i.e. as (indices of) partial recursive mappings from labels to values. Record updating is then easy to interpret, namely as the change of such a mapping for one of its inputs.

To reduce notational clutter, we assume that the set of labels is $\mathbb{N}$. A model could be given based on an arbitrary enumeration of the labels, but having natural numbers as labels saves us some irrelevant and confusing indexing of labels.

To interpret terms we first erase all their type information:

Definition 13. The type erasure $\mathit{Erase}(M)$ of a term M is defined by

$$
\begin{aligned}
\mathit{Erase}(x) &= x \\
\mathit{Erase}(\lambda x{:}\sigma.\, M) &= \lambda x.\, \mathit{Erase}(M) \\
\mathit{Erase}(MN) &= \mathit{Erase}(M)\mathit{Erase}(N) \\
\mathit{Erase}(\lambda\alpha.\, M) &= \mathit{Erase}(M) \\
\mathit{Erase}(\lambda\alpha \sqsubseteq \sigma.\, M) &= \mathit{Erase}(M) \\
\mathit{Erase}(M\sigma) &= \mathit{Erase}(M) \\
\mathit{Erase}(\langle l_1 = M_1, \ldots, l_n = M_n\rangle) &= \langle l_1 = \mathit{Erase}(M_1), \ldots, l_n = \mathit{Erase}(M_n)\rangle \\
\mathit{Erase}(M.l) &= \mathit{Erase}(M).l \\
\mathit{Erase}(M\langle l := N\rangle) &= \mathit{Erase}(M)\langle l := \mathit{Erase}(N)\rangle
\end{aligned}
$$

Definition 14. 1. If $E(x)$ is a partial recursive description of a natural number depending on some input x, we write $\boldsymbol{\lambda} x.\, E(x)$ for the index of the partial recursive function for which $\boldsymbol{\lambda} x.\, E(x)\cdot n = E(n)$.

2. If $\{l_1 \mapsto m_1, \ldots, l_n \mapsto m_n\}$ is a partial recursive mapping on natural numbers, we write $\langle\!\langle l_1 \mapsto m_1, \ldots, l_n \mapsto m_n \rangle\!\rangle$ for the index of a partial recursive function for which $\langle\!\langle l_1 \mapsto m_1, \ldots, l_n \mapsto m_n \rangle\!\rangle \cdot l_i = m_i$ for all $l_i \in \{l_1 \ldots, l_n\}$.
3. For $m, n, l \in \mathbb{N}$ we write $m\langle\!\langle l \mapsto n \rangle\!\rangle$ for the index of the partial recursive function such that

$$
m\langle\!\langle l \mapsto n \rangle\!\rangle \cdot i = \begin{cases} m \cdot i & \text{if } i \neq l, \\ n & \text{if } i \neq l. \end{cases}
$$

The constructions above are used to interpret lambda-abstractions, records, and record-updates:

Definition 15. Let η be a *term environment*, i.e. a mapping from term variables to $\mathbb{N}$. The (possibly undefined) interpretation $[M]_\eta \in \mathbb{N}$ of an *erased* term M in η is given by:

$$
\begin{aligned}
[x]_\eta &= \eta(x) \\
[\lambda x.\, M]_\eta &= \boldsymbol{\lambda} n.\ [M]_{\eta[x \mapsto n]} \\
[MN]_\eta &= [M]_\eta \cdot [N]_\eta \\
[\langle l_1 = M_1, \ldots, l_n = M_n\rangle]_\eta &= \langle\!\langle l_1 \mapsto [M_1]_\eta, \ldots, l_n \mapsto [M_n]_\eta \rangle\!\rangle \\
[M.l]_\eta &= [M]_\eta \cdot l \\
[M\langle l := N\rangle]_\eta &= [M]_\eta \langle\!\langle l \mapsto [N]_\eta \rangle\!\rangle
\end{aligned}
$$

The (possibly undefined) interpretation $[\![M]\!]_\eta \in \mathbb{N}$ of a *typed* term M in η is now defined by $[\![M]\!]_\eta = [\mathit{Erase}(M)]_\eta$.

Before it can be proved that $[\![M]\!]_\eta$ is defined for well-typed terms M, we first have to define the interpretation of types.

3.2 The Interpretation of Types

Function types are interpreted as usual, and record types as in [BL90]:

Definition 16. Let $R, S \in \mathrm{PER}$. Then $R \twoheadrightarrow S \in \mathrm{PER}$ is defined by

$$R \twoheadrightarrow S = \{(f, f') \mid \forall r, r'. (r, r') \in R \Rightarrow (f \cdot r, f' \cdot r') \in S\}.$$

Definition 17. Let $L \subseteq \mathbb{N}$ and $R_l \in \mathrm{PER}$ for every $l \in L$. Then $\langle\!\langle l \mapsto R_l \mid l \in L \rangle\!\rangle \in \mathrm{PER}$ is defined by

$$\langle\!\langle l \mapsto R_l \mid l \in L \rangle\!\rangle = \{(x, y) \mid \forall l \in L. (x \cdot l, y \cdot l) \in R_l\}.$$

We write $\langle\!\langle l_1 \mapsto R_1, \ldots, l_n \mapsto R_n \rangle\!\rangle$ for $\langle\!\langle l_i \mapsto R_i \mid l_i \in \{l_1, \ldots, l_n\} \rangle\!\rangle$. Note that $\langle\!\langle l_1 \mapsto R_1, \ldots, l_n \mapsto R_n \rangle\!\rangle = \emptyset$ as soon as one of the R_i is $\emptyset$.

To define the interpretation of types we need a suitable relation $\sqsubseteq$ on PER to interpret width-subtyping. For this we use the following operations:

Definition 18. Let $R \in \mathrm{PER}$ and $l \in \mathbb{N}$. Then

1. *R has an l-field* – written $R{\downarrow}l$ – iff $\forall x \in dom_R. x \cdot l{\downarrow}$.
2. $R \cdot l$ is the relation $\{(x \cdot l, x' \cdot l) \mid (x, x') \in R\}$.

N.B. note that $R \cdot l$ is *not* necessarily a per!

Lemma 19. *Let $R = \langle\!\langle l \mapsto R_l \mid l \in L \rangle\!\rangle \neq \emptyset$ with L a decidable set (i.e. L has a partial recursive characteristic function). Then*

1. *$R{\downarrow}l \iff l \in L$,*
2. *$R \cdot l = R_l$ for all $l \in L$.*

Proof. 1. ($\Leftarrow$): Let $l \in L$. It follows from $(r, r) \in R = \langle\!\langle l \mapsto R_l \mid l \in L \rangle\!\rangle$ that $(r \cdot l, r \cdot l) \in R_l$, and hence $r{\downarrow}l$. So $r{\downarrow}l$ for all $r \in dom_R$, i.e. $R{\downarrow}l$.
($\Rightarrow$): $R \neq \emptyset$, so we can assume an r such that $r \in dom_R$.
Suppose towards a contradiction that $R{\downarrow}l$ and $l \notin L$. Now let r' be the index of the partial recursive function with

$$r' \cdot i = \begin{cases} r \cdot i & \text{if } i \notin L, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Here the restriction to decidable sets L is needed, namely to guarantee that such an index r' exists: for decidable L the definition of r' above is partial recursive. Now $r' \in dom_R$ and $r' \cdot l{\uparrow}$, which contradicts $R{\downarrow}l$.

2. Let $l \in L$. To prove: $R \cdot l = R_l$.
($\subseteq$): Suppose $(n, n') \in R \cdot l$. Then there is an $(r, r') \in R$ such that $r \cdot l = n$ and $r' \cdot l = n'$. Since $R = \langle\!\langle l_1 \mapsto R_1, \ldots, l_n \mapsto R_n \rangle\!\rangle$ it follows from $(r, r') \in R$ that $(r \cdot l, r' \cdot l) \in R_l$, i.e. $(n, n') \in R_l$.
($\supseteq$): Suppose $(n, n') \in R_l$. To prove that $(n, n') \in R \cdot l$ we have to prove there exist some $(s, s') \in R$ such that $s \cdot l = n$ and $s' \cdot l = n'$. Such s and s' are easy to construct: $R \neq \emptyset$, so there exists some $(r, r') \in R$, and we can take $s = r\langle\!\langle l \mapsto n \rangle\!\rangle$ and $s' = r'\langle\!\langle l \mapsto n' \rangle\!\rangle$. $\square$

We define a collection RPER $\subseteq$ PER of "record pers":

Definition 20. $R \in$ RPER iff $R \in$ PER, $R = \langle\!\langle l \mapsto R{\cdot}l \mid R{\downarrow}l \rangle\!\rangle$, and $R{\cdot}l \in$ PER for all $R{\downarrow}l$.

Recall that $R \cdot l$ is not necessarily a per for $R \in$ PER. For $R \in$ RPER it is, provided $R{\downarrow}l$. All record types are interpreted as record pers:

Lemma 21. *Suppose* $R = \langle\!\langle l \mapsto R_l \mid l \in L \rangle\!\rangle$ *with all* $R_l \in$ PER *and* L *a decidable set. Then* $R \in$ RPER*.*

Proof. We distinguish two cases. If $R \neq \emptyset$, then by Lemma 19 $R{\downarrow}l \iff l \in L$ and $R{\cdot}l = R_l$ for all $l \in L$, and so $R{\cdot}l \in$ PER for all $R{\downarrow}l$ and $\langle\!\langle l \mapsto R{\cdot}l \mid R{\downarrow}l \rangle\!\rangle = \langle\!\langle l \mapsto R_l \mid l \in L \rangle\!\rangle = R$. If $R = \emptyset$, then $R{\downarrow}l$ and $R \cdot l = \emptyset$ for all $l \in \mathbb{N}$; clearly $\emptyset \in$ PER, and $\langle\!\langle l \mapsto \emptyset \mid l \in \mathbb{N} \rangle\!\rangle = \emptyset$. $\square$

The restriction to decidable sets L in the lemma above is of course no problem, as any record type in F^{width} will have a decidable set of labels. There is a relation on pers that corresponds to width-subtyping:

Definition 22. The relation $\sqsubseteq$ on PER is defined by

$$R \sqsubseteq S \iff R, S \in \text{RPER} \ \wedge \ (R = \emptyset \ \vee \ \forall S{\downarrow}l.\, R{\downarrow}l \wedge R{\cdot}l = S{\cdot}l).$$

Some simple properties of $\sqsubseteq$:

Lemma 23. *1.* $R \sqsubseteq S \Rightarrow R \subseteq S$*.*
2. $\sqsubseteq$ *is transitive.*
3. Suppose $R = \langle\!\langle l \mapsto R_l \mid l \in L \rangle\!\rangle$ *and* $S = \langle\!\langle l \mapsto R_l \mid l \in L' \rangle\!\rangle$*, with* $L' \subseteq L$*, and* L *and* L' *decidable sets. Then* $R \sqsubseteq S$*.*

The relation $\sqsubseteq$ on PER is used to interpret width-bounded quantification in types :

Definition 24. Let ξ be a *type environment*, i.e. a mapping from type variables to PER. The interpretation $[\![\sigma]\!]_\xi \in$ PER of a type σ in ξ is given by

$$\begin{aligned}
[\![\alpha]\!]_\xi &= \xi(\alpha) \\
[\![\sigma \to \tau]\!]_\xi &= [\![\sigma]\!]_\xi \twoheadrightarrow [\![\tau]\!]_\xi \\
[\![\langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n \rangle]\!]_\xi &= \langle\!\langle l_1 \mapsto [\![\sigma_1]\!]_\xi, \ldots, l_n \mapsto [\![\sigma_n]\!]_\xi \rangle\!\rangle \\
[\![\forall\alpha.\, \sigma]\!]_\xi &= \bigcap\nolimits_{R \in PER} [\![\sigma]\!]_{\xi[\alpha \mapsto R]} \\
[\![\forall\alpha \sqsubseteq \rho.\, \sigma]\!]_\xi &= \bigcap\nolimits_{R \sqsubseteq [\![\rho]\!]_\xi} [\![\sigma]\!]_{\xi[\alpha \mapsto R]}
\end{aligned}$$

3.3 Soundness

We now prove that the interpretation of types is sound with respect to $\sqsubseteq$, and that the interpretation of terms is sound with respect to typing, reduction, and equality.

Definition 25. Let ξ be a type environment and η a term environment. Then ξ *satisfies* Γ – written $\xi \models \Gamma$ – iff $\xi(\alpha) \sqsubseteq [\![\sigma]\!]_\xi$ for all $\alpha \sqsubseteq \sigma$ in Γ. The pair (η, ξ) *satisfies* Γ – written $(\eta, \xi) \models \Gamma$ – iff $\xi \models \Gamma$ and $\eta(x) \in dom_{[\![\sigma]\!]_\xi}$ for all $x : \sigma$ in Γ.

Theorem 26 (Soundness of Width-subtyping).
If $\Gamma \vdash \rho \sqsubseteq \sigma$ then $[\![\rho]\!]_\xi \sqsubseteq [\![\sigma]\!]_\xi$ for all $\xi \models \Gamma$.

Proof. Easy induction on the derivation of $\Gamma \vdash \rho \sqsubseteq \sigma$. For ($\sqsubseteq$-CONTEXT) we use the definition of $\xi \models \Gamma$, for ($\sqsubseteq$-TRANS) Lemma 23.2, and for ($\sqsubseteq$-WIDTH) Lemma 23.3. □

Theorem 27 (Soundness of Typing).
If $\Gamma \vdash M : \sigma$ then $([\![M]\!]_\eta, [\![M]\!]_\eta) \in [\![\sigma]\!]_\xi$ for all $(\eta, \xi) \models \Gamma$.

Proof. By induction on the derivation of $\Gamma \vdash M : \sigma$ we prove

1. there is a partial recursive $f : \mathbb{N}^k \to \mathbb{N}$ such that $f(\eta(x_1), \ldots, \eta(x_k)) = [\![M]\!]_\eta$ for all $(\eta, \xi) \models \Gamma$, where $x_1, \ldots, x_k$ are the term variables declared in Γ,
2. $([\![M]\!]_\eta, [\![M]\!]_{\eta'}) \in [\![\sigma]\!]_\xi$ for all $\xi \models \Gamma$ and $(\eta, \eta') \in [\![\Gamma]\!]_\xi$,

where $[\![\Gamma]\!]_\xi$ is the partial equivalence relation on term environments defined by

$$(\eta, \eta') \in [\![\Gamma]\!]_\xi \iff \forall (x{:}\sigma) \in \Gamma.\, (\eta(x), \eta'(x)) \in [\![\sigma]\!]_\xi\,.$$

(So, if $\xi \models \Gamma$ and $(\eta, \eta') \in [\![\Gamma]\!]_\xi$ then $(\eta, \xi) \models \Gamma$.)

Compared with the proof for system F there are 6 additional cases, one for each new inference rule. For the rule ($\sqsubseteq$-SUB) the property $R \sqsubseteq S \Rightarrow R \subseteq S$ (Lemma 23.1) is needed. We only treat the most interesting case:

Suppose the last step in the derivation of $\Gamma \vdash M : \sigma$ is

$$\frac{\Gamma \vdash M' : \sigma \quad \Gamma \vdash N : \tau \quad \Gamma \vdash \sigma \sqsubseteq \langle l{:}\tau \rangle}{\Gamma \vdash M'\langle l {:=} N \rangle : \sigma}\,(\text{UPDATE})$$

So $M = M'\langle l {:=} N \rangle$.

1. Follows directly from the induction hypothesis.
2. Suppose $(\eta, \eta') \in [\![\Gamma]\!]_\xi$. Define $m = [\![M]\!]_\eta$, $m' = [\![M]\!]_{\eta'}$, $S = [\![\sigma]\!]_\xi$, $n = [\![N]\!]_\eta$, $n' = [\![N]\!]_{\eta'}$, and $T = [\![\tau]\!]_\xi$. By the induction hypothesis $(m, m') \in S$ and $(n, n') \in T$. By Theorem 26 it follows from $\sigma \sqsubseteq \langle l{:}\tau \rangle$ that $S \sqsubseteq \langle\!\langle l \mapsto T \rangle\!\rangle$, so $S \in$ RPER and $S \cdot l = T$. To prove: $(m\langle\!\langle l \mapsto n \rangle\!\rangle, m'\langle\!\langle l \mapsto n' \rangle\!\rangle) \in S$. Since $S \in$ RPER this is equivalent to $\forall S{\downarrow}i.\, (m\langle\!\langle l \mapsto n \rangle\!\rangle \cdot i, m'\langle\!\langle l \mapsto n' \rangle\!\rangle \cdot i) \in S \cdot i$. Suppose $S{\downarrow}i$. We distinguish two cases:

- $i \neq l$. Then $m\langle\!\langle l \mapsto n \rangle\!\rangle \cdot i = m \cdot i$ and $m'\langle\!\langle l \mapsto n' \rangle\!\rangle \cdot i = m' \cdot i$, and $(m \cdot i, m' \cdot i) \in S \cdot i$ since $(m, m') \in S$.
- $i = l$. Then $m\langle\!\langle l \mapsto n \rangle\!\rangle \cdot i = n$ and $m'\langle\!\langle l \mapsto n' \rangle\!\rangle \cdot i = n'$, and $(n, n') \in T = S \cdot l$. □

Lemma 28. *If* $[\![M]\!]_{\eta[x \mapsto [\![N]\!]_\eta]}$ *and* $[\![[N/x]M]\!]_\eta$ *are defined, then they are equal.*

Proof. Induction on the structure of M. □

Lemma 29. *If* $M \rhd_\beta M'$ *and* $[\![M]\!]_\eta$ *and* $[\![M']\!]_\eta$ *are defined, then they are equal.*

Proof. Induction on the generation of $M \rhd_\beta M'$. Apart from the congruence rules, for which the proof is trivial, there are 5 reduction rules to consider. The cases $(\lambda\alpha.\, M)\tau \rhd_\beta M$ and $(\lambda\alpha \sqsubseteq \rho.\, M)\tau \rhd_\beta [\tau/\alpha]M$ are trivial, as $\mathit{Erase}((\lambda\alpha.\, M)\tau) = \mathit{Erase}(M) = \mathit{Erase}((\lambda\alpha \sqsubseteq \rho.\, M)\tau) = \mathit{Erase}([\tau/\alpha]M)$. The case $(\lambda x{:}\sigma.\, M)N \rhd_\beta [N/x]M$ follows from the substitution lemma (Lemma 28) as usual. The two remaining cases are very simple: it follows directly from the definition of $[\![\ ]\!]$ that

$$[\![\langle l_1 = M_1, \ldots, l_n = M_n\rangle.l_i]\!]_\eta = [\![M_i]\!]_\eta$$

$$[\![M\langle l := N\rangle.l']\!]_\eta = \begin{cases} [\![M]\!]_\eta \cdot l' & \text{if } l \neq l' \\ [\![N]\!]_\eta & \text{if } l = l' \end{cases} \qquad \square$$

Soundness of reduction easily follows from the lemma above:

Theorem 30 (Soundness of β-Reduction).
Suppose $\Gamma \vdash M \rhd_\beta M' : \sigma$. *Then* $([\![M]\!]_\eta, [\![M']\!]_\eta) \in [\![\sigma]\!]_\xi$ *for all* $(\eta, \xi) \models \Gamma$.

Proof. By soundness of typing (Theorem 27) $[\![M]\!]_\eta$ and $[\![M']\!]_\eta$ are defined and in $\mathit{dom}_{[\![\sigma]\!]_\xi}$. So by Lemma 29 $[\![M]\!]_\eta = [\![M']\!]_\eta$, and $([\![M]\!]_\eta, [\![M']\!]_\eta) \in [\![\sigma]\!]_\xi$ follows from the fact that $[\![\sigma]\!]_\xi$ is reflexive on $\mathit{dom}_{[\![\sigma]\!]_\xi}$. □

Theorem 31 (Soundness of $\beta\eta$-Equality).
Suppose $\Gamma \vdash M =_{\beta\eta} M' : \sigma$. *Then* $([\![M]\!]_\eta, [\![M']\!]_\eta) \in [\![\sigma]\!]_\xi$ *for all* $(\eta, \xi) \models \Gamma$.

Proof. Induction on the derivation of $\Gamma \vdash M =_{\beta\eta} M' : \sigma$. For the case that $M \rhd_\beta M'$ we use soundness of reduction (Theorem 30). We treat just one of the more interesting cases:

Suppose the last step in the derivation is $\dfrac{\Gamma \vdash M\langle l := M.l\rangle : \sigma \quad \text{(i)}}{\Gamma \vdash M\langle l := M.l\rangle =_{\beta\eta} M : \sigma}$.
To prove: $([\![M]\!]_\eta, [\![M\langle l := M.l\rangle]\!]_\eta) \in [\![\sigma]\!]_\xi$. By Lemma 6.2 it follows from (i) that $\sigma = \langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle$ and that $\Gamma \vdash M : \sigma$ (ii). Now

$$\begin{array}{lll} & ([\![M]\!]_\eta, [\![M\langle l := M.l\rangle]\!]_\eta) \in [\![\langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle]\!]_\xi & \\ \iff & \forall l_i \in \{l_1, \ldots, l_n\}.\, ([\![M]\!]_\eta \cdot l_i, [\![M\langle l := M.l\rangle]\!]_\eta \cdot l_i) \in [\![\sigma_i]\!]_\xi & \text{by def. } [\![\ ]\!]_\xi \\ \iff & \forall l_i \in \{l_1, \ldots, l_n\}.\, ([\![M]\!]_\eta \cdot l_i, [\![M]\!]_\eta \cdot l_i) \in [\![\sigma_i]\!]_\xi & \text{by def. } [\![\ ]\!]_\eta \\ \iff & ([\![M]\!]_\eta, [\![M]\!]_\eta) \in [\![\langle l_1{:}\sigma_1, \ldots, l_n{:}\sigma_n\rangle]\!]_\xi & \text{by def. } [\![\ ]\!]_\xi \end{array}$$

and this follows from (ii) by soundness of typing (Theorem 27). □

In the remainder of this section we show that the model provides exactly the polymorphic update operations one expects. First we show that a polymorphic update $g : (\forall\alpha \sqsubseteq \langle l{:}\sigma\rangle .\, \alpha \to \alpha)$ can only change the l-field of its input, and leaves any other fields unchanged.

Lemma 32. *Let* $(g, g) \in \bigcap_{X \sqsubseteq \langle\!\langle l \mapsto S\rangle\!\rangle} X \twoheadrightarrow X$ *and* $(m, m) \in X \sqsubseteq \langle\!\langle l \mapsto S\rangle\!\rangle$ *for some per* S. *Then* $g{\cdot}m{\cdot}i = m{\cdot}i$ *for all* $i \neq l$ *such that* $X{\downarrow}i$.

Proof. $X \sqsubseteq \langle\!\langle l \mapsto S\rangle\!\rangle$, so $X = \langle\!\langle X{\cdot}i \mid i \in X{\downarrow}i\rangle\!\rangle$ and $X{\downarrow}l$ with $X{\cdot}l = S$.
Define $Y = \langle\!\langle i \mapsto Y_i \mid X{\downarrow}i\rangle\!\rangle$ with $Y_i = \begin{cases} S & \text{if } i = l, \\ \{(m{\cdot}i, m{\cdot}i)\} & \text{if } i \neq l. \end{cases}$
Informally, Y is the record per X with all fields except l restricted to a one-point per. Clearly, $(m, m) \in Y$. Also, $Y \sqsubseteq \langle\!\langle l \mapsto S\rangle\!\rangle$, and hence $(g, g) \in Y \twoheadrightarrow Y$ and $(g{\cdot}m, g{\cdot}m) \in Y$. But by the definition of Y this means that $(g{\cdot}m{\cdot}i, g{\cdot}m{\cdot}i) \in Y_i$ for all $i \in I$, and so $(g{\cdot}m{\cdot}i, g{\cdot}m{\cdot}i) \in \{(m{\cdot}i, m{\cdot}i)\}$ for all $i \neq l$ such that $X{\downarrow}i$. □

An immediate consequence of this lemma:

Corollary 33. *If* $\Gamma \vdash g : (\forall\alpha \sqsubseteq \langle l{:}\sigma\rangle .\, \alpha \to \alpha)$ *and* $\Gamma \vdash M : \tau$ *with* $\Gamma \vdash \tau \sqsubseteq \langle l{:}\sigma\rangle$, *then* $[\![(g\tau M).l_i]\!]_\eta = [\![M.l_i]\!]_\eta$ *for all* $\Gamma \vdash M.l_i : \tau_i$ *with* $l_i \neq l$ *and* $\eta \models \Gamma$.

The type $(\forall\alpha \sqsubseteq \langle l{:}\sigma\rangle .\, \alpha \to \alpha)$ contains at least one member for every function $f : \langle l{:}\sigma\rangle \to \langle l{:}\sigma\rangle$, namely $\lambda\alpha \sqsubseteq \langle l{:}\sigma\rangle .\, \lambda x{:}\alpha .\, x\langle l := (fx).l\rangle$. In fact, it is difficult to imagine functions of this type that are not of this form. The mapping from $\langle l{:}\sigma\rangle \to \langle l{:}\sigma\rangle$ to $(\forall\alpha \sqsubseteq \langle l{:}\sigma\rangle .\, \alpha \to \alpha)$ given above is indeed an isomorphism in the PER model[1] :

Lemma 34. $\mathbb{N}/\,[\![\forall\alpha \sqsubseteq \langle l{:}\sigma\rangle .\, \alpha \to \alpha]\!]_\xi$ *and* $\mathbb{N}/\,[\![\langle l{:}\sigma\rangle \to \langle l{:}\sigma\rangle]\!]_\xi$ *are isomorphic for all* $\xi \models \Gamma$.

Proof. Let $\rho = (\forall\alpha \sqsubseteq \langle l{:}\sigma\rangle .\, \alpha \to \alpha)$, $S = [\![\langle l{:}\sigma\rangle]\!]_\xi$, and $R = [\![\rho]\!]_\xi$. The isomorphism between $\mathbb{N}/R$ and $\mathbb{N}/(S \twoheadrightarrow S)$ is given by the interpretations of

$$\begin{aligned} \phi &= \lambda g \in \rho .\, g\langle l{:}\sigma\rangle && : \rho \to \langle l{:}\sigma\rangle \to \langle l{:}\sigma\rangle \\ \psi &= \lambda f \in \langle l{:}\sigma\rangle \to \langle l{:}\sigma\rangle .\, \lambda\alpha \sqsubseteq \sigma .\, \lambda x{:}\alpha .\, x\langle l := (fx).l\rangle && : (\langle l{:}\sigma\rangle \to \langle l{:}\sigma\rangle) \to \rho \end{aligned}$$

i.e. by $[\![\phi]\!] = \boldsymbol{\lambda} x.\, x$ and $[\![\psi]\!] = \boldsymbol{\lambda} f \boldsymbol{\lambda} x \boldsymbol{\lambda} i.\, \begin{cases} x{\cdot}i & \text{if } i \neq l \\ f{\cdot}x{\cdot}l & \text{if } i = l \end{cases}$
Let $\Phi \in \mathbb{N}/R \to \mathbb{N}/(S \twoheadrightarrow S)$ and $\Psi \in \mathbb{N}/(S \twoheadrightarrow S) \to \mathbb{N}/R$ be the functions on equivalence classes induced by $[\![\phi]\!]$ and $[\![\psi]\!]$. So $\Phi([g]_R) = [[\![\phi]\!] \cdot g]_{S \twoheadrightarrow S}$ and $\Psi([f]_{S \twoheadrightarrow S}) = [[\![\psi]\!]{\cdot}f]_R$, where $[n]_X$ denotes the X-equivalence class containing n. It follows from soundness of typing (Theorem 27) that Φ and Ψ are well-defined functions on equivalence classes. That they are each other's inverses follows from the properties

[1] Note that interpretations of types are isomorphic if there is an isomorphism between their equivalence classes, as $[\![\sigma]\!]_\xi$ gives the notion of equality for interpretations of terms of type σ, and so the number of different interpretations of terms of type σ is the number of $[\![\sigma]\!]_\xi$-equivalence classes.

1. $(f, f) \in S \twoheadrightarrow S \;\Rightarrow\; (f, [\![\phi]\!] \cdot [\![\psi]\!] \cdot f) \in S \twoheadrightarrow S$,
2. $(g, g) \in R \;\Rightarrow\; (g, [\![\psi]\!] \cdot [\![\phi]\!] \cdot g) \in R$,

which are proved below. Note that $[\![\phi]\!]$ is simply the identity, so $[\![\phi]\!] \cdot [\![\psi]\!] \cdot f = [\![\psi]\!] \cdot f$ and $[\![\psi]\!] \cdot [\![\phi]\!] \cdot g = [\![\psi]\!] \cdot g$.

1. $(f, f) \in S \twoheadrightarrow S$
   $$\begin{array}{lll} & \iff \forall (x, x') \in S.\, (f \cdot x, f \cdot x') \in S & \text{by def. } \twoheadrightarrow \\ & \iff \forall (x, x') \in S.\, (f \cdot x \cdot l, f \cdot x' \cdot l) \in [\![\sigma]\!]_\xi & \text{since } S = \langle\!\langle l \mapsto [\![\sigma]\!]_\xi \rangle\!\rangle \\ & \iff \forall (x, x') \in S.\, (f \cdot x \cdot l, [\![\psi]\!] \cdot f \cdot x' \cdot l) \in [\![\sigma]\!]_\xi & \text{since } f \cdot x' \cdot l = [\![\psi]\!] \cdot f \cdot x' \cdot l \\ & \iff \forall (x, x') \in S.\, (f \cdot x, [\![\psi]\!] \cdot f \cdot x') \in S & \text{since } S = \langle\!\langle l \mapsto [\![\sigma]\!]_\xi \rangle\!\rangle \\ & \iff (f, [\![\psi]\!] \cdot f) \in S \twoheadrightarrow S & \text{by def. } \twoheadrightarrow \end{array}$$
2. Suppose $(g, g) \in R$. To prove: $(g, [\![\psi]\!] \cdot g) \in R$. Since $R = \bigcap_{X \sqsubseteq S} X \twoheadrightarrow X$, this is equivalent to $\forall X \sqsubseteq S.\, \forall (x, x') \in X.\, (g \cdot x, [\![\psi]\!] \cdot g \cdot x') \in X$.
   Let $X \sqsubseteq S$ and $(x, x') \in X$. So $X \neq \emptyset$ and it follows by the definition of $\sqsubseteq$ that $X = \langle\!\langle i \mapsto X \cdot i \mid X \downarrow i \rangle\!\rangle$ with $X \downarrow l$ and $X \cdot l = S \cdot l = [\![\sigma]\!]_\xi$. To prove: $(g \cdot x, [\![\psi]\!] \cdot g \cdot x') \in X$, which is equivalent to $\forall X \downarrow i.\, (g \cdot x \cdot i, [\![\psi]\!] \cdot g \cdot x' \cdot i) \in X \cdot i$.
   Suppose $X \downarrow i$. We distinguish two cases:
   – $i = l$. Then $[\![\psi]\!] \cdot g \cdot x' \cdot i = g \cdot x' \cdot l$, so to prove: $(g \cdot x \cdot l, g \cdot x' \cdot l) \in X \cdot l$.
     From $(g, g) \in R \supseteq X \twoheadrightarrow X$ and $(x, x') \in X$ it follows that $(g x, g x') \in X$, and so $(g \cdot x \cdot l, g \cdot x' \cdot l) \in X \cdot l$.
   – $i \neq l$. Then $[\![\psi]\!] \cdot g \cdot x' \cdot i = x' \cdot i$, so to prove: $(g \cdot x \cdot i, x' \cdot i) \in X \cdot i$.
     It follows from $(x, x') \in X$ and $X \downarrow i$ that $(x \cdot i, x' \cdot i) \in X \cdot i$. So it suffices to prove $g \cdot x \cdot i = x \cdot i$, which follows from Lemma 32. □

4 Related Work

[Oho95] also describes an extension of system F with width-bounded quantification and a primitive for record updating, but *without* subsumption. His main interest however is the predicative part of this system, in particular an ML-style (i.e. implicitly typed) type system that corresponds to this predicative part, and the problem of its compilation.

Several other extensions of F that provide polymorphic record updates have been proposed [CM91][Car92][Zwa95][HP96]. The system F^{width} is simpler than all of these. It is also less expressive, but it does provide all the record operations needed for the existential object model in [PT94].

Instead of updating, the systems in [CM91] and [Car92] provide operations for removing and adding fields to records as primitives. This has several consequences. Firstly, in order to type these primitives we need operations for removing and adding fields to record *types*, whereas in F^{width} no new operations on types are needed. Secondly, to safely add fields to records we need types that express "negative" information (i.e. tell about the absence of certain fields). In F^{width} we only need types that express "positive" information (i.e. about the presence of certain fields).

Although the system described in [CM91] is very expressive, it can not express width-subtyping or width-bounded quantifications. In this system the polymorphic update `have_birthday` will have type

$$\forall\sigma \leq \langle \mathtt{age}{:}\mathtt{Nat}\rangle.\, \sigma \to \sigma - \mathtt{age} + \langle \mathtt{age}{:}\mathtt{Nat}\rangle$$

where $-l$ and $+\langle l : \sigma\rangle$ are the operations of removing and adding fields to record types. The bounded quantification in this type can not be restricted to those types σ for which $\sigma - \mathtt{age} + \langle \mathtt{age}:\mathtt{Nat}\rangle$ will be equal to σ (i.e. to the width-subtypes of $\langle \mathtt{age}{:}\mathtt{Nat}\rangle$).

The system *F#* presented in [Zwa95] provides a "merge"-operation that can be used to concatenate a record to another record, overwriting any common fields, provided the records have "compatible" types. F^{width} is a subsystem of *F#*: the update operation is a simple case of the merge operation, and width-subtyping is a combination of ordinary subtyping and compatibility: width-subtypes are exactly the compatible subtypes.

The notion of width-subtyping is also considered in [BL94] but in quite a different setting, namely the lambda calculus with additional primitives for objects – so-called object calculus – introduced in [FHM94]. Consequently, width-subtyping is there not a relation on record types but a relation on special object types.

[AC95] describes another object calculus with a subtype relation on object types. But here the subtype relation is more general than just width-subtyping: annotation of the fields in object types controls whether depth-subtyping is allowed on each individual field, so that both conventional subtyping and width-subtyping are essentially special cases of this single subtype relation.

4.1 Comparison with Positive Subtyping

In [HP96] another restriction of subtyping is used to deal with the update-operations, namely *positive subtyping*. We write $\leq^+$ for positive subtyping, and F^{pos} for the extension of F with positive subtyping given in [HP96]. Positive subtyping is a weaker relation than width-subtyping, i.e. $\sqsubseteq\, \subseteq\, \leq^+\, \subseteq\, \leq$. For $\leq^+$ we have all the usual subtyping rules, with the exception of the contrapositive rule for function types. In particular, $\leq^+$ includes both width- and depth-subtyping. So, for example $\langle l{:}\langle x, y{:}\mathtt{Nat}\rangle\rangle \leq^+ \langle l{:}\langle x{:}\mathtt{Nat}\rangle\rangle$. A consequence is a more general update-operation. E.g. a record $M : \langle l{:}\langle x, y{:}\mathtt{Nat}\rangle\rangle$ can be updated in its l-field with $N : \langle x{:}\mathtt{Nat}\rangle$, with as result a copy of M with the x-field of its l-field updated with the x-field of N, but the y-field of its l-field unchanged. This is known as a *recursive* or *deep* update. There is a price for this more general update-operation:

- Update-operations have to be annotated with more type information in F^{pos}: the types of both M and N have to be supplied as explicit type parameters in $M\langle l := N\rangle$.
- The notion of reduction in F^{pos} is more limited than in F^{width}. Whereas $M\langle l := N\rangle.l$ reduces to N in F^{width}, in F^{pos} they might not even be equal.

(E.g. consider the example above, where $M : \langle l{:}\langle x, y{:}\mathtt{Nat}\rangle\rangle$ and $N : \langle x{:}\mathtt{Nat}\rangle$). Reduction in F^{pos} is a *typed* reduction, i.e. it depends on type information in terms, whereas reduction in F^{width} – as in F – is an *untyped* reduction.

- The PER model for F^{pos} is more complicated than the one for F^{width}. For F^{pos} it is not possible to erase all type information from terms as a first step when defining the semantics of terms.

The more general notion of subtyping and a more general update-operation of F^{pos} are not required to write classes in the sense of [PT94]: all the examples of class definitions given in [HP96] are typable in F^{width}, and all the equalities that are proved for these examples in [HP96] also hold in F^{width}. In fact, in F^{width} all these equalities are simple β-equalities.

The only serious disadvantage of F^{width} compared to F^{pos} is that because of the weakness of the subtyping relation – in particular the lack of congruence rules allowing for instance $\sigma \to \langle l{:}\rho, m{:}\rho\rangle \sqsubseteq \sigma \to \langle l{:}\rho\rangle$ – the property of minimal typing is lost. However, this property is regained when F^{width} is extended with conventional subtyping, as discussed below.

5 Further Extensions

A further extension of F^{width} needed for the object encoding of [PT94] is conventional subtyping. This is because we want $\mathtt{Object}(M')$ to be a subtype of $\mathtt{Object}(M)$ if M' a richer interface than M, and we clearly do not have $\mathtt{Object}(M') \sqsubseteq \mathtt{Object}(M)$, with $\mathtt{Object}$ as in Sect. 2.1. The positive subtyping of [HP96] suffers from the same deficiency. Extending F^{width} with conventional subtyping will result in a system with two subtyping relations, width-subtyping $\sqsubseteq$ and conventional subtyping $\leq$, with $\sqsubseteq$ contained in $\leq$. The syntax becomes more complicated, but as far as the PER semantics is concerned this extension poses no problems, since our PER model of F^{width} is compatible with the PER models of $F_{\leq}$. Both subtype relations can be interpreted in the PER world: the normal subset inclusion between relations as interpretation for $\leq$, and $\sqsubseteq$ as defined in Definition 22 as interpretation for $\sqsubseteq$.

Maybe the complexity of having two subtype relations – conventional subtyping and width-subtyping – could be avoided by distinguishing updatable and non-updatable fields in records, and then only allowing depth-subtyping on non-updatable fields, as in [AC95] [Pie96], but a model for such a system would probably be more complicated and very syntactic in flavour.

Other useful extensions would be F^{ω}-style type operators and a fixpoint operator for terms. The interpretation of F^{ω}-style type operators is not a problem in the PER model, but if a fixpoint is added the PER model we have given no longer suffices, and it remains to be seen if the more complex PER models for system F with recursion described in [AP90] [Ama91] [BM92] could be adapted.

6 Conclusions

We have presented a system F^{width} that extends system F with a primitive for updating records and width-subtyping on record types. It provides the polymorphic record-updates needed for the class definitions in the existential object model of [PT94].

The combination of width-subtyping and a primitive operation for updating seems to be the easiest way to provide polymorphic record-updates. Intuitively width-subtyping and updating are very simple notions: the rules of F^{width} are fairly obvious, the record-update has a very simple operational semantics (given by the reduction relation $\rhd_\beta$), and a straightforward interpretation in the PER model. Decomposing record-updating into more primitive operations for field-removal and record-extension, as in [CM91], results in more expressive and complex systems than F^{width}.

The main technical result is the PER model for F^{width}. Key to this model construction is the important observation that it possible to tell which pers are interpretations of record types. This enables us to give an interpretation of width-subtyping – which is a restricted form of "structural" subtyping – without having to resort to the very syntactical model constructions like those sketched in [CM91].

Width-subtyping is a restriction of positive subtyping introduced in [HP96]. As discussed in Sect. 4.1, this restriction has several advantages, notably the simpler PER model and the simpler – and untyped – reduction relation giving an operational semantics.

Acknowledgements

I want to thank Benjamin Pierce, Jan Zwanenburg, and the anonymous referees for their helpful comments on this paper.

References

[AC95] Martín Abadi and Luca Cardelli. An imperative object calculus. In P. D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *TAPSOFT'95: Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 471–485, 1995.

[AC96] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer, 1996.

[Ama91] Roberto M. Amadio. Recursion over realizability structures. *Information and Computation*, 90(2):55–85, 1991.

[AP90] Martín Abadi and Gordon Plotkin. A PER-model of polymorphism and recursive types. In *Logic in Computer Science*, pages 355–365. IEEE, 1990.

[BL90] Kim B. Bruce and Giuseppe Longo. A modest model of records, inheritance, and bounded quantification. *Information and Computation*, 87:196–240, 1990. Also in [GM94].

[BL94] Viviana Bono and Luigi Liquori. A subtyping for the Fisher-Honsell-Mitchell lambda calculus of objects. In Leszek Pacholski and Jerzy Tiuryn, editors, *CSL '94*, volume 933 of *LNCS*, pages 16–30. Springer, 1994.

[BM92] Kim B. Bruce and John C. Mitchell. PER models of subtyping, recursive types and higher-order polymorphism. In *Principles of Programming Languages*, pages 316–327. ACM, 1992.

[Car92] Luca Cardelli. Extensible records in a pure calculus of subtyping. Research report 81, DEC Systems Research Center, 1992. Also in [GM94].

[CM91] Luca Cardelli and John Mitchell. Operations on records. *Mathematical Structures in Computer Science*, 1:3–48, 1991. Also in [GM94].

[CW85] Luca Cardelli and Peter Wegner. On understanding types, data abstraction and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.

[FHM94] Kathleen Fisher, Furio Honsell, and John C. Mitchell. A lambda calculus of objects and method specialization. *Nordic Journal of Computing*, 1(1):3–37, 1994.

[FM94] Kathleen Fisher and John C. Mitchell. Notes on typed object-oriented programming. In *Proceedings of Theoretical Aspects of Computer Software (TACS'94) , Sendai, Japan*, volume 789 of *LNCS*, pages 844–886. Springer, 1994.

[GM94] Carl A. Gunter and John C. Mitchell. *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design.* The MIT Press, 1994.

[HP96] Martin Hofmann and Benjamin C. Pierce. Positive subtyping. *Information and Computation*, 126(1):11–33, 10 April 1996.

[Oho95] Atsushi Ohori. A polymorphic record calculus and its compilation. *ACM Trans. on Prog. Lang. and Syst.*, 17(6):845–895, 1995.

[Pie96] Benjamin C. Pierce. Even simpler type-theoretic foundations for object-oriented programming. manuscript, March 1996.

[PT94] Benjamin C. Pierce and David N. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, April 1994.

[Zwa95] Jan Zwanenburg. Record concatenation with intersection types. Computing Science Report (95/34), Eindhoven University of Technology, 1995.