Parametricity as Subtyping

(Preliminary Report)

QingMing Ma*
School of Computer Science
Carnegie-Mellon University
Qingming.Ma@cs.cmu.edu

Abstract

A polymorphic function is parametric if it has uniform behavior for all type parameters. This property is useful when writing, reasoning about, and compiling functional programs.

We show how to syntactically define and reason about parametricity in a language with intersection types and bounded polymorphism. Within this framework, parametricity is subtyping, and reasoning about parametricity becomes reasoning about the well-typedness of terms. This work also demonstrates the expressiveness of languages that combine intersection types and bounded polymorphism.

1 Introduction

A polymorphic function is parametric if it uses the same algorithm regardless of which type parameter is instantiated. As a consequence, it has a uniform behavior over all type parameters, in the sense that for any related inputs, the function produces related outputs.

Let us look at an example. Consider the polymorphic "doubling" function

$$\mathsf{double} = \Lambda\tau.\ \lambda f_{\tau\to\tau}.\ \lambda x_\tau.\ f(fx) : \forall\tau.\ (\tau\to\tau)\to(\tau\to\tau)\ .$$

By passing **int** and **bool** to this function, we get the doubling functions for **int** and **bool**:

$$\mathsf{double}_{\mathbf{int}} = \lambda f_{\mathbf{int}\to\mathbf{int}}.\ \lambda x_{\mathbf{int}}.\ f(fx) : (\mathbf{int}\to\mathbf{int})\to(\mathbf{int}\to\mathbf{int})$$

$$\mathsf{double}_{\mathbf{bool}} = \lambda f_{\mathbf{bool}\to\mathbf{bool}}.\ \lambda x_{\mathbf{bool}}.\ f(fx) : (\mathbf{bool}\to\mathbf{bool})\to(\mathbf{bool}\to\mathbf{bool})$$

Although $\mathsf{double}_{\mathbf{int}}$ and $\mathsf{double}_{\mathbf{bool}}$ accept arguments of different types, they have the same algorithm. Namely, for any inputs f and x, the computation of double applies the f twice to the x. Assume that we represent **bool** by **int** in such a way that even numbers represent true, and odd numbers represent false. That is, we have a relation $R \subseteq \mathbf{int}\times\mathbf{bool}$ such that

$$R = \{\langle 2n, \mathsf{true}\rangle\} \cup \{\langle 2n+1, \mathsf{false}\rangle\}\ .$$

Given $g{:}\,\mathbf{int}\to\mathbf{int}$ and $h{:}\,\mathbf{bool}\to\mathbf{bool}$. The function g is said to represent h iff

$$\langle n, b\rangle \in R \quad \text{implies} \quad \langle gn, hb\rangle \in R\ ,$$

or, g relates h by $R{\Rightarrow}R$. Parametricity of double implies that, if g represents h, then $\mathsf{double}_{\mathbf{int}}g{:}\,\mathbf{int}\to\mathbf{int}$ represents $\mathsf{double}_{\mathbf{bool}}h{:}\,\mathbf{bool}\to\mathbf{bool}$ as well, *i.e.*,

$$\langle n, b\rangle \in R \text{ implies } \langle \mathsf{double}_{\mathbf{int}}gn, \mathsf{double}_{\mathbf{bool}}hb\rangle \in R\ .$$

This can be checked out straightforwardly, since $\mathsf{double}_{\mathbf{int}}gn = g(gn)$ and $\mathsf{double}_{\mathbf{bool}}hb = h(hb)$.

If we let $\mathsf{even?}{:}\,\mathbf{int}\to\mathbf{bool}$ be the representation function that tests whether an integer is even or odd, then that $g{:}\,\mathbf{int}\to\mathbf{int}$ represents $h{:}\,\mathbf{bool}\to\mathbf{bool}$ is equivalent to saying that g and h satisfy the equation

$$\mathsf{even?}\circ g = h\circ\mathsf{even?}\ .$$

Parametricity of double implies that $\mathsf{double}_{\mathbf{int}}g$ and $\mathsf{double}_{\mathbf{bool}}h$ satisfy the same property:

$$\mathsf{even?}\circ(\mathsf{double}_{\mathbf{int}}g) = (\mathsf{double}_{\mathbf{bool}}h)\circ\mathsf{even?}\ .$$

In fact, this property holds not just for the function double but for all functions of the type $\forall\tau.(\tau\to\tau)\to(\tau\to\tau)$.

In other words, for any polymorphic type (e.g. $\forall\tau.(\tau\to\tau)\to(\tau\to\tau)$), there is a property (or condition), called parametricity, which is determined by the structure of the type and possessed by all parametric polymorphic functions of this type. The property is not

*This research was supported in part by NSF Grant CCR-8922109 and in part by the Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division, U.S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597.

derivable from rules for computing polymorphic functions (e.g. $\beta\eta$-rules). Nevertheless, it is useful when writing, reasoning about, and compiling functional programs. This property is possessed by the polymorphic functions in the languages such as `Fun` [7], `Quest` [5], and `Leap` [18], where polymorphic functions are "first-class values", as well as by the polymorphic functions in `ML`-like languages, where polymorphic functions are not "first-class values".

At this point, one may ask: How can one systematically reason about parametricity so that one needn't prove "free theorems" ([27]) over and over again? How can one show that a given polymorphic function satisfies the parametricity condition of its type?

In this paper, we propose a syntactic method to define and reason about parametricity. We study the parametricity of the Girard-Reynolds system F in a language called **Target**, which is a variant of Pierce's system $F_\wedge$ ([20]) that contains intersection types and bounded polymorphism. The parametricity condition for each polymorphic type in F is represented as a subtyping relation, and reasoning about parametricity becomes a simple matter of reasoning about the well typedness of terms in **Target**. We also extend this method to studying parametricity of $F_\wedge$.

To make the point clear, we first briefly describe intersection types and bounded polymorphism. Intuitively, intersection types are kinds of finite polymorphism, and bounded polymorphism is polymorphism where type parameters may be limited to a certain range of types.

A term e has an intersection type $\sigma_1 \wedge \sigma_2$ if it has both type σ_1 and type σ_2; and a term e has a bounded polymorphic type $\forall\tau{\leq}\sigma_1.\ \sigma_2$, if e has the type σ_2 under the assumption $\tau \leq \sigma_1$ of subtyping. A polymorphic function e of type $\forall\tau{\leq}\sigma_1.\ \sigma_2$ can only be instantiated to those type parameters that are subtypes of σ_1.

We view a relation R between two types σ' and σ'' as a subtyping relation $\mathcal{S}(R) \leq \sigma'{\times}\sigma''$ in **Target**. For each polymorphic type $\forall\tau.\sigma$ in F, we have a type $(\forall\tau.\ \sigma)^*$ and a subtyping relation

$$(\forall\tau.\ \sigma)^* \leq (\forall\tau.\ \sigma){\times}(\forall\tau.\ \sigma)$$

in **Target**. The parametricity condition for $\forall\tau.\ \sigma$ is determined by the subtyping in the sense that a polymorphic function p of the type $\forall\tau.\ \sigma$ is parametric iff the typing judgement

$$\vdash \langle p, p\rangle : (\forall\tau.\ \sigma)^*$$

is valid in **Target**. We show that all polymorphic functions defined in F have this property.

Let us go back to the above example. For the type $\forall\tau.\sigma = \forall\tau.(\tau{\to}\tau){\to}(\tau{\to}\tau)$, the type $(\forall\tau.\ \sigma)^*$ is defined to be

$$(\forall\tau.\ \sigma)^* = \forall\tau'\forall\tau''\forall\tau{\leq}\tau'{\times}\tau''.\ (\theta{\to}\theta){\wedge}(\sigma'{\times}\sigma'')$$

where $\theta = (\tau{\to}\tau) \wedge ((\tau'{\to}\tau'){\times}(\tau''{\to}\tau''))$, $\sigma' = [\tau'/\tau]\sigma$, and $\sigma'' = [\tau''/\tau]\sigma$.

Assume we have established that all polymorphic functions in F satisfy the parametricity conditions of their respective types. Then we have the following valid typing judgement in **Target**:

$$\vdash \langle \mathsf{double},\ \mathsf{double}\rangle : (\forall\tau.\ \sigma)^*\ .$$

By substituting τ' and τ'' with **int** and **bool** and τ with the type $\mathcal{S}(R)$ determined by the relation $R \subseteq \mathbf{int}{\times}\mathbf{bool}$ and applying a typing rule for intersection types, we have the following typing judgement:

$$\begin{aligned}\vdash\ & \langle \mathsf{double}_{\mathbf{int}}, \mathsf{double}_{\mathbf{bool}}\rangle :\\ & ((\mathcal{S}(R){\to}\mathcal{S}(R)) \wedge ((\mathbf{int}{\to}\mathbf{int}){\times}(\mathbf{bool}{\to}\mathbf{bool})))\\ & {\to}((\mathcal{S}(R){\to}\mathcal{S}(R)) \wedge ((\mathbf{int}{\to}\mathbf{int}){\times}(\mathbf{bool}{\to}\mathbf{bool})))\ .\end{aligned}$$

Hence, for $g : \mathbf{int}{\to}\mathbf{int}$ and $h : \mathbf{bool}{\to}\mathbf{bool}$, if

$$\vdash \langle g, h\rangle : (\mathcal{S}(R){\to}\mathcal{S}(R)) \wedge ((\mathbf{int}{\to}\mathbf{int}){\times}(\mathbf{bool}{\to}\mathbf{bool}))$$

i.e., $\langle f, g\rangle$ maps pairs related by R to pairs related by R, then

$$\begin{aligned}\vdash\ & \langle \mathsf{double}_{\mathbf{int}} f, \mathsf{double}_{\mathbf{bool}} g\rangle :\\ & (\mathcal{S}(R){\to}\mathcal{S}(R)) \wedge ((\mathbf{int}{\to}\mathbf{int}){\times}(\mathbf{bool}{\to}\mathbf{bool}))\ .\end{aligned}$$

i.e., $\langle \mathsf{double}_{\mathbf{int}} f, \mathsf{double}_{\mathbf{bool}} g\rangle$ also maps pairs related by R to pairs related by R. Hence, the property for $\mathsf{double}_{\mathbf{int}}$ and $\mathsf{double}_{\mathbf{bool}}$ we discussed above is derived from reasoning about the well typedness.

The study of parametricity was initiated by Reynolds in [24], where parametricity conditions were defined in terms of naive set-theoretic models by using second-order logical relations. Parametricity has received much attention in recent years. Here is a brief summary:

- Parametricity has been further studied by Mitchell and Meyer [17], Bainbridge, Freyd, Scedrov, and Scott [3], Freyd [11], Abramsky, Mitchell, Scedrov, and Wadler [2], Abramsky and Jensen [1], Cardelli, Mitchell, Martini, and Scedrov [6], and Ma and Reynolds [15]. The focus of this research is on model-theoretic aspects of parametricity, such as representation independence, observational equivalence, strictness analysis, constructing parametric models.

- Wadler [27] gave many interesting examples showing how to get "free theorems" in functional programming and program verification by using parametricity. Since second-order logical relations can be formalized in second-order logic systems, this leads to Mairson's ([16]) proof-theoretic study of parametricity.

- Parametricity introduces equalities between functions that are not implied by usual computation rules (e.g. β-η rules). Using these equalities properly, the computation of a polymorphic function may be substantially simplified, and also more efficient code may be generated by a compiler. Pfenning [19] pointed out the possible connection between parametricity and uniqueness of the representation of a term in an inductive type.

However, to date, parametricity has been mostly studied at the model level. (Mairson's work is an exception. We'll discuss this later.) Therefore, it is not easily understood by people who are unfamiliar with model theory. This is because, for the polymorphic languages, the models themselves are already difficult to understand. Hence, it is desirable to have a syntactic treatment of parametricity, so that it can be easily understood and convenient to reason about.

The paper is organized as follows. In section 2, we introduce the source languages F and $F_\wedge$ and the target language **Target**. We study parametricity of F in Section 3. In section 4, some examples are given to show that Walder's "free theorems" can be recast in out framework. In section 5, we discuss some model-theoretic issues of our method and give some initial results about parametricity of PER models. In section 6, we extend our method to studying parametricity of the system $F_\wedge$.

2 Languages

In this section, we introduce briefly three languages: F, $F_\wedge$, and **Target**. Both F and $F_\wedge$ are source languages. We study parametricity of F and $F_\wedge$ by translating them into **Target**.

2.1 The system F

The Girard-Reynolds [13, 23] system F provides a type-theoretic formalization of parametric polymorphism by adding to the ordinary typed lambda calculus the ability to pass types as parameters.

The set of F types are defined by

$$\sigma ::= \tau \mid \sigma_1 \to \sigma_2 \mid \forall \tau.\sigma$$

The set of unchecked F terms are defined by:

$$e ::= x \mid \lambda x{:}\sigma.\ e \mid e_2 e_1 \mid \Lambda \tau.\ e \mid e[\sigma]$$

A context Γ is a finite (possibly empty) set of variables with associated types that are assigned by Γ:

$$\Gamma = \{x_1 : \sigma_1, \cdots, x_n : \sigma_n\}$$

such that no variable occurs twice. Typing judgements have the from

$$\Gamma \vdash e : \sigma\ ,$$

which means that the term e has type σ under the context Γ. The valid typing judgements in F are those that are provable from the following typing rules:

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma}\ (\text{VAR}) \qquad \frac{\Gamma \cup \{x : \sigma\} \vdash e : \rho}{\Gamma \vdash \lambda x{:}\sigma\ e : \sigma \to \rho}\ (\text{ABS})$$

$$\frac{\Gamma \vdash e_1 : \sigma \quad \Gamma \vdash e_2 : \sigma \to \rho}{\Gamma \vdash e_2 e_1 : \rho}\ (\text{AP})$$

$$\frac{\Gamma \vdash e : \sigma \quad \tau \notin \Gamma}{\Gamma \vdash \Lambda \tau.\ e : \forall \tau.\sigma}\ (\text{T}{-}\text{ABS}) \qquad \frac{\Gamma \vdash e : \forall \tau.\ \sigma}{\Gamma \vdash e[\alpha] : [\alpha/\tau]\sigma}\ (\text{T}{-}\text{AP})$$

We skip the equational rules (e.g. β and η-rules). The details can be found in [13, 23].

The system introduced is an explicitly typed version of F An implicit F can be got by erasing type information in e of $\Gamma \vdash e : \sigma$ above:

$$\begin{aligned} \text{Erase}(x) &= x \\ \text{Erase}(\lambda x{:}\sigma.\ e) &= \lambda x.\ \text{Erase}(e) \\ \text{Erase}(e_2 e_1) &= \text{Erase}(e_2)\text{Erase}(e_1) \\ \text{Erase}(\Lambda \tau.\ e) &= \text{Erase}(e) \\ \text{Erase}(e[\sigma]) &= \text{Erase}(e) \end{aligned}$$

For any type derivation for $\Gamma \vdash e : \sigma$ in the explicit F, there is a corresponding type derivation for $\Gamma \vdash \text{Erase}(e) : \sigma$ in the implicit F. However, we note that, in explicit F, type derivation is unique for a well-typed term. But this is no longer true in implicit F.

2.2 The system $F_\wedge$

The system $F_\wedge$, introduced in Pierce's dissertation [21], is a calculus that combines intersection types ([25]) and bounded polymorphism ([7]). The system presented here is an implicitly typed version. More information about syntactic, semantic, and pragmatic details can be found in [21]. Information about intersection types or bounded polymorphism can be found in [4, 22, 25, 26] or [6, 7, 8, 9, 10, 12, 20].

The set of $F_\wedge$ types is defined by:

$$\sigma ::= \tau \mid \mathbf{ns} \mid \sigma_1 \to \sigma_2 \mid \forall \tau \leq \sigma_1.\ \sigma_2 \mid \sigma_1 \wedge \sigma_2$$

Here $\wedge$ is the binary intersection operator, **ns** denotes the null intersection, and $\forall \tau \leq \sigma_1.\ \sigma_2$ is for bounded polymorphism. We denote $\forall \tau \leq \mathbf{ns}.\ \sigma_2$ as $\forall \tau.\ \sigma_2$.

A type context is a sequence

$$\Theta = \tau_1 \leq \sigma_1; \cdots; \tau_n \leq \sigma_n$$

of type variables with associated bounds, such that no type variable τ is defined twice, and no defined type variable appears in its own bound or in the bounds of type variables defined to its left.

Subtyping judgements are of the form:

$$\Theta \vdash \sigma_1 \leq \sigma_2\ ,$$

which means that the subtyping relation $\sigma_1 \leq \sigma_2$ holds under the type context Θ. The valid subtyping judgements in $F_\wedge$ are those that are provable from the subtyping rules listed in the Appendix.

Term contexts of $F_\wedge$ are defined recursively as follows:

$$\Gamma ::= \{\} \mid \Gamma; \tau \leq \sigma \mid \Gamma; x : \sigma$$

such that $\hat{\Gamma}$, got by erasing the declarations of the form $x{:}\sigma$, is a type context.

Typing judgements in $F_\wedge$ are of the form

$$\Gamma \vdash e : \theta \, .$$

Equality judgements in $F_\wedge$ are of the form

$$\Gamma \vdash e' = e'' : \theta \, .$$

Subtyping rules, typing rules, and the equational theory of $F_\wedge$ appear in the Appendix.

2.3 Target

Target is a variant of Pierce's $F_\wedge$. First, product types $(\sigma \times \rho)$ and related subtyping rules are added in order to study binary "relations". Accordingly, the new terms of the forms $\langle e', e'' \rangle$, $\mathrm{fst}(e)$, and $\mathrm{snd}(e)$ are added. This addition is unnecessary if we just want to study unary "relations".

Second, some extra subtyping and typing rules are used. These special rules are listed in the Appendix. For parametricity with unary relations, only two extra rules are needed. They are PRE-DEF and Q-LB. The rule DIST-IP is used for studying parametricity of $F_\wedge$, otherwise can be omitted.

Third, **Target** is an open language in the sense that it is parametrized by predefined types and predefined subtying relations between closed types. This makes it possible that, for different applications, **Target** may subscribe different predefined types and predefined subtyping relations.

Also, for the special purpose of this paper, we have the form $\langle e', e'' \rangle : \sigma$ of typing, when σ is a subtype of a product type $\sigma' \times \sigma''$. It denotes a term of σ that is coerced to $\langle e', e'' \rangle$ by the coercion function from σ to $\sigma' \times \sigma''$.

3 Parametricity of F

We study parametricity of F and explain informally why our method works.

3.1 Parametricity

We study parametricity of the system F. For each type σ in F, we define a type σ^* in **Target**. For each type derivation $\Gamma \vdash e{:}\sigma$ in the explicit F, we have a corresponding type derivation for $\Gamma \vdash \mathrm{Erase}(e){:}\sigma$ in the implicit F. We translate the latter type derivation to a type derivation for $\Gamma^* \vdash \langle \mathrm{Erase}(e),\ \mathrm{Erase}(e) \rangle {:} \sigma^*$ in **Target**, whereby the parametricity for e can be reasoned about. In the following discussion, we neglect the erasing procedure from the explicit F to the implicit one, and treat all type derivations as in the implicit F.

Let σ be an arbitrary type in F with free type variables $\{\tau_1, \ldots, \tau_n\}$. Then σ is also a type in **Target**. Let Θ be the type context

$$\tau_1' \leq \mathbf{ns}; \tau_1'' \leq \mathbf{ns}; \tau_1 \leq \tau_1' \times \tau_1'', \ldots, \tau_n' \leq \mathbf{ns}; \tau_n'' \leq \mathbf{ns}; \tau_n \leq \tau_n' \times \tau_n'',$$

and let $\sigma' = [\tau_1'/\tau_1, \ldots, \tau_n'/\tau_n]\sigma$ and $\sigma'' = [\tau_1''/\tau_1, \ldots, \tau_n''/\tau_n]\sigma$, where all τ_i' and τ_i'' are fresh type variables in **Target**. Note that, if σ is a closed type, then $\sigma' = \sigma'' = \sigma$.

We inductively define, for each type σ in F, a type σ^* in **Target** as follows:

$$\begin{aligned} \tau^* &= \tau \, , \\ (\sigma \rightarrow \rho)^* &= \sigma^* \rightarrow \rho^* \wedge ((\sigma' \rightarrow \rho') \times (\sigma'' \rightarrow \rho'')) \, , \\ (\forall \tau.\ \sigma)^* &= \forall \tau'. \forall \tau''. \forall \tau \leq \tau' \times \tau''.\ \sigma^* \, . \end{aligned}$$

Lemma 3.1 *In* **Target**, *the following subtyping judgement is valid:*

$$\Theta \vdash \sigma^* \leq \sigma' \times \sigma'' \, .$$

Definition 3.1 *Let $\forall \tau.\ \sigma$ be a closed type. A polymorphic function of the type $\forall \tau.\ \sigma$ is parametric iff*

$$\vdash \langle p, p \rangle : (\forall \tau.\ \sigma)^*$$

is valid in **Target**.

We now prove that all closed terms of polymorphic types in F are parametric. Following the work in [24], we prove a more general result, *i.e.*, the Abstraction Theorem.

Let Γ be the context $\{x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k\}$. Notice that we have

$$\Theta \vdash \sigma_i^* \leq \sigma_i' \times \sigma_i'' \qquad (1 \leq i \leq k) \, .$$

We define a context

$$\Gamma^* = \Theta; \langle x_1', x_1'' \rangle {:} \sigma_1^*, \ldots, \langle x_n', x_n'' \rangle {:} \sigma_k^* \, ,$$

in **Target**, where all x_i' and x_i'' are fresh variables in **Target**. Foe each term e in F, we define

$$\begin{aligned} e' &= [x_n'/x_n, \ldots, x_1'/x_1]e \, , \\ e'' &= [x_n''/x_n, \ldots, x_1''/x_1]e \, . \end{aligned}$$

Then, for each derivation $\Gamma \vdash e : \sigma$ in F, we define a type derivation

$$\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$$

in **Target** by induction on the typing inference rule used in the last step in deriving the typing judgement in F.

- If $\Gamma \vdash e : \sigma$ is got by using the rule (VAR), then $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ is got from

$$\frac{\langle x', x'' \rangle : \sigma^* \in \Gamma^*}{\Gamma^* \vdash \langle x', x'' \rangle : \sigma^*}$$

- If $\Gamma \vdash e : \sigma$ is got by using the rule (ABS), then $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ is got from

$$\frac{\Gamma^*; \langle x', x'' \rangle : \sigma^* \vdash \langle e', e'' \rangle : \rho^*}{\Gamma^* \vdash \langle \lambda x'.\ e', \lambda x''.\ e'' \rangle : (\sigma \rightarrow \rho)^*}$$

  by the rule ARROW-I'.

- If $\Gamma \vdash e : \sigma$ is got by using the rule (AP), then $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ is got from

$$\frac{\Gamma^* \vdash \langle e'_1, e''_1 \rangle : \sigma^* \quad \Gamma^* \vdash \langle e'_2, e''_2 \rangle : (\sigma \rightarrow \rho)^*}{\Gamma^* \vdash \langle e'_2 e'_1, e''_2 e''_1 \rangle : \rho^*}$$

  by the rule ARROW-E'.

- If $\Gamma \vdash e : \sigma$ is got by using the rule (T-ABS), then $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ is got from

$$\frac{\Gamma^*; \tau' \leq \mathbf{ns}; \tau'' \leq \mathbf{ns}; \tau \leq \tau' \times \tau'' \vdash \langle e', e'' \rangle : \sigma^*}{\Gamma^* \vdash \langle e', e'' \rangle : \forall \tau' \forall \tau'' \forall \tau \leq \tau' \times \tau''.\ \sigma^*}$$

- If $\Gamma \vdash e : \sigma$ is got by using the rule (T-AP), then $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ is got from

$$\frac{\Gamma^* \vdash \langle e', e'' \rangle : (\forall \tau.\ \sigma)^* \quad \hat{\Gamma^*} \vdash \alpha^* \leq \alpha' \times \alpha''}{\Gamma^* \vdash \langle e', e'' \rangle : [\alpha^*/\tau][\alpha'/\tau', \alpha''/\tau'']\sigma^*}$$

Lemma 3.2 *For any types σ and α in F, we have*

$$([\alpha/\tau]\sigma)^* = [\alpha^*/\tau][\alpha'/\tau', \alpha''/\tau'']\sigma^* .$$

Proposition 3.1 [Abstraction Theorem] *If $\Gamma \vdash e : \sigma$ is a valid typing judgement in F, then*

$$\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$$

in **Target**

Note that each valid typing judgement in F is a valid typing judgement in **Target** as well. So, the Abstraction Theorem is equivalent to saying that

$$\frac{\Gamma \vdash e : \sigma}{\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*}$$

is a valid typing rule in **Target** for all valid typing judgements $\Gamma \vdash e : \sigma$ in F.

Corollary 3.1 [Parametricity] *Let $\forall \tau.\ \sigma$ be a closed type, and $\vdash e : \sigma$ be valid in F. Then $\vdash \langle e', e'' \rangle : (\forall \tau.\sigma)^*$ is valid in* **Target**.

The above proving process provides us a syntactic way to use parametricity by viewing the parametricity condition of a polymorphic type $\forall \tau.\sigma$ as the subtyping

$$\Theta \vdash (\forall \tau.\sigma)^* \leq (\forall \tau.\sigma) \times (\forall \tau.\sigma) .$$

To reason about parametricity of a term $\Gamma \vdash e : \forall \tau.\ \sigma$, we use the valid typing judgement:

$$\Gamma^* \vdash \langle e', e'' \rangle : (\forall \tau.\ \sigma)^* .$$

As we noted before, **Target** may subscribe different predefined types and subtyping relations. Accordingly, the parametricity conditions defined may have different meanings. For a particular application, we may define **Target** with a particularly choose set of predefined types and subtyping relations.

For example, we may define the set of predefined types and subtyping relations such that, for any two closed types α' and α'', and any subset R of closed terms of $\alpha' \times \alpha''$, *i.e.*, there is a constant type $\mathcal{S}(R)$ and a predefined subtyping relation $\vdash \mathcal{S}(R) \leq_{\text{pre}} \alpha' \times \alpha''$, such that for $\vdash e' : \alpha'$ and $\vdash e'' : \alpha''$ in F

$$\vdash \langle e', e'' \rangle : \mathcal{S}(R) \text{ iff } \langle e', e'' \rangle \in R .$$

In many cases, we do not want to distinguish between terms that are provably equal in F. Hence, every predefined type $\mathcal{S}(R)$ should be closed terms that are provably equal in F. This property is preserved by the construction of σ^*. We only consider such cases in this paper.

3.2 Relations as subtyping

Parametricity is originally defined in [24] by second-order logical relations. As shown in [15, 2] to define second-order logical relations, one first need to have a set $\mathcal{R}$ of relations between closed types. This $\mathcal{R}$ may be chosen differently for different applications. But $\mathcal{R}$ should be closed for the relation operators $\Rightarrow$ and $\forall$ defined below.

Let $R^{\sigma', \sigma''}$ and $R^{\rho', \rho''}$ be two relations. A relation $R^{\sigma', \sigma''} \Rightarrow R^{\rho', \rho''}$ over types $\sigma' \rightarrow \rho'$ and $\sigma'' \rightarrow \rho''$ is defined such that, for $\Gamma \vdash f' : \sigma' \rightarrow \rho'$ and $\Gamma \vdash f'' : \sigma'' \rightarrow \rho''$,

$$\langle f', f'' \rangle \in R^{\sigma', \sigma''} \Rightarrow R^{\rho', \rho''}$$
$$\text{iff}$$
$$(\forall \langle e', e'' \rangle \in R^{\sigma', \sigma''})\ \langle f'e', f''e'' \rangle \in R^{\rho', \rho''} .$$

To define relations on a variable type σ, let us assume that σ contain only one free type variable τ. Since the τ in σ may be substituted by different type parameters, a relation $R^{\sigma, \sigma}$ on σ should change along with different relations between type parameters. Hence, $R^{\sigma, \sigma}$ should contain a relation variable R^{τ}. By substituting R^{τ} with $R^{\alpha', \alpha''}$, we get a relation $[R^{\alpha', \alpha''}/R^{\tau}]R^{\sigma}$ between $[\alpha'/\tau]\sigma$ and $[\alpha''/\tau]\sigma$.

The relation $\forall R^{\tau}.\ R^{\sigma,\sigma}$ on $\forall\tau.\ \sigma$ is defined such that, for any $\vdash p' : \forall\tau.\ \sigma$ and $\vdash p'' : \forall\tau.\ \sigma$,

$$\langle p', p''\rangle \in \forall R^{\tau}.\ R^{\sigma}$$
iff
$$(\forall R^{\alpha',\alpha''} \in \mathcal{R})\ \langle p'[\alpha'], p''[\alpha'']\rangle \in [R^{\alpha',\alpha''}/R^{\tau}]R^{\sigma}\ .$$

If σ contains $n{+}1$ free type variables, then $n{+}1$ relation variables are used, and the relation operator $\forall$ is defined pointwise by fixing n relation variables to n relations at a time.

By reviewing each relation $R^{\sigma',\sigma''}$ on closed type σ' and σ'' as a subtyping relation

$$\vdash \mathcal{S}(R^{\sigma',\sigma''}) \leq \sigma'\times\sigma''$$

such that $\mathcal{S}(R^{\sigma',\sigma''})$ has all elements that are related by $R^{\sigma',\sigma''}$, the relation operators $\Rightarrow$ and $\forall$ can be properly represented by intersection and bounded quantification constructors.

Let σ' and σ'' be two types in F, which contains type variables $\tau_1, \ldots, \tau_n$. Any relation $R^{\sigma',\sigma''}$ between σ' and σ'' should contain n relation variables. Hence, the subtyping corresponding to $R^{\sigma',\sigma''}$ is of the form

$$\Theta \vdash \mathcal{S}(R^{\sigma',\sigma''}) \leq \sigma'\times\sigma''\ ,$$

where Θ is as in Section 3.

Assume that we have established subtyping relations

$$\Theta \vdash \mathcal{S}(R^{\sigma',\sigma''}) \leq \sigma'\times\sigma''\ \text{ and }\ \Theta \vdash \mathcal{S}(R^{\rho',\rho''}) \leq \rho'\times\rho''\ .$$

Then the subtyping

$$\Theta \vdash \mathcal{S}(R^{\sigma',\sigma''}\Rightarrow R^{\rho',\rho''}) \leq (\sigma'\to\rho')\times(\sigma''\to\rho'')\ ,$$

should be such that, for $\Gamma \vdash f' : \sigma'\to\rho'$ and $\Gamma \vdash f'' : \sigma''\to\rho''$,

$$\Gamma^* \vdash \langle f', f''\rangle : \mathcal{S}(R^{\sigma',\sigma''}\Rightarrow R^{\rho',\rho''})$$
iff
$$\Gamma^* \vdash \langle e', e''\rangle : \mathcal{S}(R^{\sigma',\sigma''}) \text{ implies } \Gamma^* \vdash \langle f'e', f''e''\rangle : \mathcal{S}(R^{\rho',\rho''})\ .$$

Hence, we define $\mathcal{S}(R^{\sigma',\sigma''}\Rightarrow R^{\rho',\rho''})$ to be

$$(\mathcal{S}(R^{\sigma',\sigma''})\to\mathcal{S}(R^{\rho',\rho''}))\wedge((\sigma'\to\rho')\times(\sigma''\to\rho''))\ .$$

To represent the relation $\forall R^{\tau}.\ R^{\sigma}$, let σ' and σ'' contain $n{+}1$ type variables $\tau_1, \ldots, \tau_n, \tau$. Suppose we have established (where $\Theta' = \Theta; \tau'\leq\mathbf{ns}; \tau''\leq\mathbf{ns}; \tau\leq\tau'\times\tau''$)

$$\Theta' \vdash \mathcal{S}(R^{\sigma',\sigma''}) \leq [\tau'/\tau]\sigma'\times[\tau''/\tau]\sigma''\ .$$

We want to have a subtyping relation

$$\Theta \vdash \mathcal{S}(R^{\forall\tau.\ \sigma',\forall\tau.\ \sigma''}) \leq \forall\tau.\ \sigma'\times\forall\tau.\ \sigma''$$

such that, for any $\Gamma \vdash p' : \forall\tau.\ \sigma'$ and $\Gamma \vdash p'' : \forall\tau.\ \sigma''$,

$$\Gamma^* \vdash \langle p', p''\rangle : \mathcal{S}(R^{\forall\tau.\ \sigma',\forall\tau.\ \sigma''})$$
iff
$$(\forall\alpha',\alpha'')(\forall\alpha)\Theta \vdash \alpha \leq \alpha'\times\alpha'')$$
$$\Gamma^* \vdash \langle p'[\alpha'], p''[\alpha'']\rangle : [\alpha/\tau, \alpha'/\tau', \alpha''/\tau'']\mathcal{S}(R^{\sigma',\sigma''})\ .$$

This leads to the definition

$$\mathcal{S}(\forall R^{\tau}.\ R^{\sigma',\sigma''}) \stackrel{\text{def}}{=} \forall\tau'.\forall\tau''.\forall\tau\leq\tau'\times\tau''.\ \mathcal{S}(R^{\sigma',\sigma''})\ .$$

4 Examples

Some examples are given to show that Wadler's "free theorems" can be recast in our framework.

4.1 Identity

Consider the polymorphic type $\forall\tau.\tau\to\tau$. For any closed term $\vdash e : \forall\tau.\tau\to\tau$ in F, let $p = \text{Erase}(e)$. Hence,

$$\vdash \langle p, p\rangle : \forall\tau'\forall\tau''\forall\tau\leq\tau'\times\tau''.\ \tau\to\tau \wedge (\tau'\to\tau')\times(\tau''\to\tau'')$$

in **Target**. For any two closed types α' and α'' in F and any subtyping $\vdash \beta\leq\alpha'\times\alpha''$ in **Target**,

$$\vdash \langle p, p\rangle : \beta\to\beta \wedge (\alpha'\to\alpha')\times(\alpha''\to\alpha'')\ .$$

Thus, for $\vdash \langle b', b''\rangle : \alpha'\times\alpha''$,

$$\vdash \langle b', b''\rangle : \beta \text{ implies } \vdash \langle pb', pb''\rangle : \beta\ .$$

Specifically, if $\langle b', b''\rangle : \beta$ is defined by a function $f : \alpha'\to\alpha''$ such that $b'' = fb'$, then $p(fb') = f(pb'')$. Filling in the type information in these terms, the equality means $e[\alpha''](fb') = f(e[\alpha']b')$.

If $(\forall b' \in \alpha')\ f(b') = x$, then $e[\alpha']x = x$. Hence, every closed term of $\forall\tau.\ \tau\to\tau$ is extensionally equivalent to the polymorphic identity function $\Lambda\tau.\lambda x{:}\tau.\ x$.

4.2 An isomorphism

It is well known ([11, 24]) that, with parametricity, the type $\beta = \forall\tau.\ (\alpha\to\tau)\to\tau$ is isomorphic to α in any extensional model. To get such an isomorphism, let

$$\begin{aligned} i &\equiv \lambda f{:}\beta.\ f[\alpha](\lambda x{:}\alpha.\ x) : \beta\to\alpha\ ,\\ j &\equiv \lambda z{:}\alpha.\ \Lambda\tau.\lambda g{:}\alpha\to\tau.\ gz : \alpha\to\beta\ . \end{aligned}$$

It is easy to check that the equality $i(j(z)) = z$ holds in F. However, the term

$$j(i(f)) = \Lambda\tau.\lambda g{:}\alpha\to\tau.\ g(f[\alpha](\lambda x{:}\alpha.\ x))$$

is not $\beta\eta$-equivalence to f. To be the case, it must have

$$f[\tau]g = g(f[\alpha](\lambda x{:}\alpha.\ x))\ .$$

By erasing type information, we need to show that for any type τ

$$f{:}\beta, g{:}\alpha\to\tau \vdash fg = g(f(\lambda x.\ x)) : \tau\ .$$

(As an aside, if one side of the equality is used for another in a compiler, the object code efficiency may be increased.)

The parametricity condition for β (treating α as a type variable) is the subtyping

$$\alpha' \leq \mathbf{ns}; \alpha'' \leq \mathbf{ns}; \alpha \leq \alpha' \times \alpha'' \vdash \beta^* \leq \beta' \times \beta'' .$$

By instantiating α' and α'' to the closed type α and $\alpha \leq \alpha' \times \alpha''$ to $\mathcal{I}(\alpha) \leq \alpha \times \alpha$, we have

$$\vdash \langle f, f \rangle : ([\mathcal{I}(\alpha)/\alpha][\alpha/\alpha', \alpha/\alpha'']\beta^*$$

where $\vdash \mathcal{I}(\alpha) \leq \alpha \times \alpha$ is the subtyping such that, for closed terms e' and e'' of F, $\vdash \langle e', e'' \rangle : \mathcal{I}(\alpha)$ iff $\vdash e' = e'' : \alpha$ in F.

For any closed types θ and $\vdash g : \alpha \to \theta$ in F, let $R \leq \alpha \times \theta$ be the subtyping that, for any $\vdash e' : \alpha$ and $\vdash e'' : \theta$ in F, $\vdash \langle e', e'' \rangle : R$ iff $\vdash ge' = e'' : \theta$ in F. Hence, for any $\vdash f : \beta$, we have:

$$\vdash \langle f, f \rangle : [(\mathcal{I}(\alpha) \to R) \wedge (\alpha \to \alpha) \times (\alpha \to \theta)] \to R .$$

Since $\vdash \langle \lambda x.\ x,\ g \rangle : \mathcal{I}(\alpha) \to R \wedge ((\alpha \to \alpha) \times (\alpha \to \theta))$, we have

$$\vdash \langle f(\lambda x.\ x), fg \rangle : R .$$

So, $fg = g(f(\lambda x.\ x))$. By filling in type information, this means $f[\theta]g = g(f[\alpha](\lambda x{:}\alpha.\ x))$. With extensionality, we have $f[\tau]g = g(f[\alpha](\lambda x{:}\alpha.\ x))$, and therefore $\langle i, j \rangle$ is an isomorphic pair.

4.3 Products

The product of two types α and β in F is defined as $\forall\tau.(\alpha \to \beta \to \tau) \to \tau$ (tentatively denoted as $\alpha \bullet \beta$ to avoid the confusion with $\alpha \times \beta$ in **Target**). The pair composed of $x{:}\alpha$ and $y{:}\beta$ is represented as $\Lambda\tau.\lambda f{:}\alpha \to \beta \to \tau.\ fxy$. By erasing type information, we have

$$x{:}\alpha, y{:}\beta \vdash \lambda f.fxy : \alpha \bullet \beta$$

(denoted as $[x, y]$). (A similar argument as in section 4.2 can be made to show that, with parametricity, $\alpha \bullet \beta$ is the usual Cartesian product. We will not get into the details, but assume that the fact is true.)

Using the method of section 3, we have the subtyping

$$\alpha \leq \alpha' \times \alpha''; \beta \leq \beta' \times \beta'' \vdash (\alpha \bullet \beta)^* \leq (\alpha' \bullet \beta') \times (\alpha'' \bullet \beta'') .$$

We show that

$$\Gamma \vdash \langle \lambda f'.\ f'x'y', \lambda f''.\ f''x''y'' \rangle : (\alpha \bullet \beta)^*$$
$$\text{iff}$$
$$\langle x', x'' \rangle : \alpha \ \text{ and } \ \langle y', y'' \rangle : \beta$$

where $\Gamma = x'{:}\alpha'; x''{:}\alpha''; y'{:}\beta'; y''{:}\beta''$. In other words, $[x', x'']$ and $[y', y'']$ are related componentwise.

One direction can be got directly from the proof of the Abstraction Theorem:

$$(x{:}\alpha, y{:}\beta)^* \vdash \langle \lambda f'.f'x'y',\ \lambda f''.f''x''y'' \rangle : (\alpha \bullet \beta)^* .$$

To get the other direction, we need to use the parametricity of the polymorphic projections:

$$\vdash \mathbf{p}_1 = \lambda p.\ p(\lambda x.\lambda y.x) : \forall\alpha\forall\beta(\alpha \bullet \beta \to \alpha)$$
$$\vdash \mathbf{p}_2 = \lambda p.\ p(\lambda x.\lambda y.y) : \forall\alpha\forall\beta(\alpha \bullet \beta \to \beta)$$

For $\mathbf{p}_1$, it means that the typing

$$\alpha \leq \alpha' \times \alpha''; \beta \leq \beta' \times \beta'' \vdash \langle \mathbf{p}_1,\ \mathbf{p}_1 \rangle : (\alpha \bullet \beta \to \alpha)^*$$

is valid. If

$$\langle x', x'' \rangle : \alpha' \times \alpha''; \langle y', y'' \rangle : \beta' \times \beta'' \vdash \langle [x', y'], [x'', y''] \rangle : (\alpha \bullet \beta)^*$$

then we have

$$(x{:}\alpha; y{:}\beta)^* \vdash \langle \mathbf{p}_1[x', y'], \mathbf{p}_2[x'', y''] \rangle : \alpha$$

Since $\mathbf{p}_1[x', y'] = x'$ and $\mathbf{p}_1[x'', y''] = x''$ in F, thus $\langle x', x'' \rangle : \alpha$. Using a similar procedure, it can be shown that $\langle y', y'' \rangle : \beta$.

Let α', β', α'', and β'' are closed types, and $g{:}\alpha' \to \alpha''$ and $h{:}\alpha'' \to \beta''$ be two functions. The functions g and h determine two subtyping relations $\alpha \leq \alpha' \times \alpha''$ and $\beta \leq \beta' \times \beta''$. We can define a function $g \bullet h : \alpha' \bullet \beta' \to \alpha'' \bullet \beta''$ that maps $[x', y']$ to $[x'', y'']$. This function determines a subtyping relation $(\alpha \bullet \beta)^* \leq (\alpha' \bullet \beta') \times (\alpha'' \bullet \beta'')$. From the parametricity of projections, we have

$$g \circ \mathbf{p}_1 = \mathbf{p}_1 \circ g \bullet h \ \text{ and } \ h \circ \mathbf{p}_2 = \mathbf{p}_2 \circ g \bullet h .$$

By filling in type information, it means

$$\begin{aligned} g \circ \mathbf{p}_1[\alpha'][\beta'] &= \mathbf{p}_1[\alpha''][\beta''] \circ g \bullet h \\ h \circ \mathbf{p}_2[\alpha'][\beta'] &= \mathbf{p}_2[\alpha''][\beta''] \circ g \bullet h \end{aligned}$$

4.4 Lists

Most "free theorems" in [27] are given for functions on lists. Due to the space limitation, we can not recast them here. However, we note that there is no essential difficult to do so.

In F, $\mathsf{list}(\alpha)$ is defined as $\forall\tau.(\alpha \to \tau \to \tau) \to \tau \to \tau$. A list $[x_1, \ldots, x_n]$ of n elements is coded as $\Lambda\tau.\lambda f{:}\alpha \to \tau \to \tau.\lambda a{:}\tau.fx_1(fx_2 \ldots (fx_n a) \ldots)$, or $\lambda f.fx_1(fx_2 \ldots (fx_n a) \ldots)$, without type information.

Similar to the discussion for the case of products, for any subtyping $\alpha \leq \alpha' \times \alpha''$, we may build a subtyping relation $\mathsf{list}(\alpha)^* \leq \mathsf{list}(\alpha') \times \mathsf{list}(\alpha'')$, such that for two lists $[x'_1, \ldots, x'_n]$ and $[x''_1, \ldots, x''_n]$ of the same length n,

$$\langle x'_i, x''_i \rangle : \alpha' \times \alpha'' \vdash [x'_1, \ldots, x'_n], [x''_1, \ldots, x''_n]] : \mathsf{list}(\alpha)^*$$
$$\text{iff}$$
$$(\forall i \leq n) \langle x'_i, x''_i \rangle : \alpha$$

(To prove this, in addition to parametricity, induction is needed.) With this property of the subtyping $\mathsf{list}(\alpha)^* \leq \mathsf{list}(\alpha') \times \mathsf{list}(\alpha'')$, it is not difficult to rebuild Wadler's "free theorems".

5 Semantics

We address briefly some semantic issues, sketch the possibility of using our method to study parametricity of models of F, and give some initial results about parametricity of PER.

5.1 PER interpretation

Models of subtyping and bounded quantification have received much attention recently. The most successful models to date are constructed from partial equivalence relations (abbreviated p.e.r.'s) on natural numbers ω or an applicative structure $\langle D, \cdot \rangle$. In these models, subtyping is interpreted as sub-relations (*i.e.*, subsets by viewing relations as subsets of $\omega \times \omega$ or $D \times D$).

However, the subtyping rule DIST-PA of **Target** is not valid in PER models. Hence, we limit our discussion to parametricity with unary relations, since no product types and related subtyping are needed in such cases. **Target** just contains two more subtyping reules (PRE-DEF and Q-LB) than $F_\wedge$, and therefore can be modeled by PER.

Let $\langle D, \cdot \rangle$ be an applicative structure, such that $D \cong D \rightarrow D$. A p.e.r. A on D is a symmetric and transitive relation on D. We use $\langle a, b \rangle \in A$ or aAb when a relates b by A. Let $\mathsf{Q}(A) = \{\{a\}_A \mid \langle a, a \rangle \in A\}$ denote the quotient set of A, where $\{a\}_A = \{b \mid aAb\}$.

A p.e.r. B is said to be a sub-p.e.r. of C (denoted as $B \leq C$) iff, for any d and e, dBe implies dCe.

To model type constructors in **Target**, we define:

$$\begin{aligned} A \Rightarrow B &= \{\langle d_1, d_2 \rangle \mid (\forall e_1 A e_2)\ (d_1 \cdot e_1) B (d_2 \cdot e_2)\}\ , \\ \mathrm{NS} &= D \times D\ , \\ A \cap B &= \{\langle d_1, d_2 \rangle \mid d_1 A d_2 \ \text{ and } \ d_1 B d_2\}\ , \\ \cap_{B \leq C} \mathcal{F}(B) &= \{\langle d_1, d_2 \rangle \mid (\forall B \leq C)\ d_1 \mathcal{F}(B) d_2\}\ . \end{aligned}$$

where $\mathcal{F}$ is a function from PER to PER.

With the above definition, it is easy to see how to define types and subtyping relations of **Target** in PER, if the predefined types and subtyping relations are properly defined. It is also easy to check that all subtyping rules are sound.

5.2 Parametric models

Given a model of F and second-order logical relations on the model. Informally, the model is parametric with respect to the logical relations if, for any closed type $\forall\tau.\ \sigma$, $[\![\forall\tau.\ \sigma]\!]$ contains only parametric functions with respect to the given logical relations. In other words, if $f \in [\![\forall\tau.\ \sigma]\!]$ then, for any α' and α'' in the model, $f[\alpha']$ and $f[\alpha'']$ map related arguments to related values in the model.

Assume we have a model of **Target** that extends the model of F in three ways: Every type of F is interpreted the same as in the model of F; The set of relations defined on the model of F are the same as the set of subtyping relations in the extended model; For every type σ in F, $[\![\sigma^*]\!]$ coincides with the relation on $[\![\sigma]\!]$ defined in the model of F. Then the model of F is parametric if, for every closed types $\forall\tau.\ \sigma$, $[\![\forall\tau.\ \sigma]\!]$ is isomorphic to $[\![(\forall\tau.\ \sigma)^*]\!]$ under the coercion function in the model of **Target**.

Let us consider PER models of F. The extension of PER to **Target** failed to satisfy the above conditions. The reason is that if we define unary relations as sub-p.e.r.'s, the coincidence of $[\![\sigma^*]\!]$ and R^σ can not be achieved. To save the situation, we need to change the definition for $[\![\sigma \wedge \rho]\!]$. But this may require the change of the definition for subtyping and bounded polymorphism, which is beyond the scope of this paper.

Hence, we consider the case that a unary relation R^A on a p.e.r. A is a subset of $\mathsf{Q}(A)$. It is clear that R^A determines a unique sub-p.e.r. of A. Thus, subtyping relations in PER contain unary relations as instances.

Let R^A and R^B be two relations on A and B, respectively. The relation $R^A \Rightarrow R^B$ is the subset

$$\{\{d\}_{A \Rightarrow B} \mid \{a\}_A \in R^A \text{ implies } \{d \cdot a\}_B \in R^B\}$$

of $\mathsf{Q}(A \Rightarrow B)$. This set is exactly the quotient set of the p.e.r. (by viewing R^A and R^B as p.e.r.'s)

$$(R^A \Rightarrow R^B) \wedge (A \Rightarrow B)\ .$$

Let σ be a type in F with only one type variable τ and with no occurrence of type binding $\forall$ inside. The relation $R^{\forall\tau.\ \sigma}$ defined on $\forall\tau.\ \sigma$ is the subset

$$\{\{d\}_{[\![\forall\tau.\ \sigma]\!]} \mid (\forall A \in \mathrm{PER})(\forall R^A)\{d\}_{[\![\sigma]\!](A)} \in \mathsf{Q}([\![\sigma^*]\!](A, R^A))\}$$

of $\mathsf{Q}([\![\forall\tau.\ \sigma]\!])$, which is the quotient set of the p.e.r.

$$\cap_{A \in \mathrm{PER}} \cap_{R^A} [\![\sigma^*]\!](A, R^A).$$

Hence, $[\![(\forall\tau.\ \sigma)^*]\!] = \cap_{A \in \mathrm{PER}} \cap_{B \leq A} \sigma^*(A, B)$ is a sub-p.e.r. of $R^{\forall\tau.\ \sigma}$ (when viewed as a p.e.r.). We have:

$$[\![(\forall\tau.\ \sigma)^*]\!] \leq R^{\forall\tau.\ \sigma} \leq [\![\forall\tau.\ \sigma]\!]\ .$$

Especially, for the type of the form

$$\forall\tau.\ \sigma = \forall\tau.\ \tau \rightarrow \tau \rightarrow \cdots \rightarrow \tau$$

in F, $(\forall\tau.\ \sigma)^*$ (for unary relations) is of the form

$$\forall\tau' \forall\tau \leq \tau'.\ (\tau \rightarrow \tau \rightarrow \cdots \rightarrow \tau) \wedge (\tau' \rightarrow \tau' \rightarrow \cdots \rightarrow \tau')\ .$$

Obviously, $\forall\tau.\ \sigma$ and $(\forall\tau.\ \sigma)^*$ are interpreted by the same p.e.r. in PER.

This implies that, in PER, $\forall\tau.\ \tau \rightarrow \tau \rightarrow \cdots \rightarrow \tau$ contains only parametric (w.r.t. unary subset relations) functions.

6 Parametricity of $F_\wedge$

In this section, we show how our method can be extended to studying parametricity of $F_\wedge$.

Let σ be an arbitrary type in $F_\wedge$ with free type variables $\{\tau_1, \ldots, \tau_n\}$. Let σ' and σ'' be the same as before. The type σ^* in **Target** as follows:

$$\begin{array}{rcl} \mathbf{ns}^* & = & \mathbf{ns}\ , \\ \tau^* & = & \tau\ , \\ (\sigma \wedge \rho)^* & = & \sigma^* \wedge \rho^*\ , \\ (\sigma \to \rho)^* & = & \sigma^* \to \rho^* \wedge ((\sigma' \to \rho') \times (\sigma'' \to \rho''))\ , \\ (\forall \tau \leq \rho.\ \sigma)^* & = & \forall \tau' \leq \rho'. \forall \tau'' \leq \rho''. \forall \tau \leq \rho^* \wedge (\tau' \times \tau'').\ \sigma^*\ . \end{array}$$

In object-oriented programming, properties of a program are often verified by using the properties of its supertypes. This is called "supertype abstraction" (see [14]). In our framework, it means that if $\vdash \sigma \leq \rho$ is valid in $F_\wedge$, then $\vdash \sigma^* \leq \rho^*$ must be valid in **Target**. This condition is also needed to prove the Abstraction Theorem for $F_\wedge$.

Let Θ be a type context in $F_\wedge$. A type context Θ^* in **Target** can be inductively defined by

- $\{\}^* = \{\}$;
- $(\Theta; \tau \leq \sigma)^* = \Theta^*; \tau' \leq \sigma'; \tau'' \leq \sigma''; \tau \leq \sigma^* \wedge (\tau' \times \tau'')$.

Proposition 6.1 *If $\Theta \vdash \sigma \leq \rho$ is valid in $F_\wedge$, then both $\Theta^* \vdash \sigma^* \leq \sigma' \times \sigma''$ and $\Theta^* \vdash \sigma^* \leq \rho^*$ are valid in* **Target**.

Lemma 6.1 *For any well defined type σ of $F_\wedge$, we have*

$$([\alpha/\tau]\sigma)^* = [\alpha^*/\tau][\alpha'/\tau', \alpha''/\tau'']\sigma^*\ .$$

To prove a similar abstraction theorem for $F_\wedge$, for any context Γ in $F_\wedge$, a context Γ^* in **Target** is defined as:

- $\{\}^* = \{\}$;
- $(\Gamma; \tau \leq \sigma)^* = \Gamma^*; \tau' \leq \sigma'; \tau'' \leq \sigma''; \tau \leq \sigma^* \wedge (\tau' \times \tau'')$;
- $(\Gamma; x : \sigma)^* = \Gamma^*; \langle x', x'' \rangle : \sigma^*$.

It is obvious that $\hat{\Gamma}^* = \hat{\Gamma^*}$. For each typing derivation $\Gamma \vdash e : \sigma$ in $F_\wedge$, a typing derivation $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ in **Target** can be defined. The interesting cases are:

- If $\Gamma \vdash e : \sigma_2$ is got by using the rule All-I, then $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ is got from

$$\frac{\Gamma^*; \tau' \leq \sigma_1'; \tau'' \leq \sigma_1''; \tau \leq \sigma_1^* \wedge (\tau' \times \tau'') \vdash \langle e', e'' \rangle : \sigma_2^*}{\Gamma^* \vdash \langle e', e'' \rangle : (\forall \tau \leq \sigma_1.\ \sigma_2)^*}$$

- If $\Gamma \vdash e : \sigma$ is got by using the rule All-E, then $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ is got from

$$\frac{\Gamma^* \vdash \langle e', e'' \rangle : (\forall \tau \leq \sigma_1.\ \sigma_2)^* \quad \hat{\Gamma}^* \vdash \bar{\alpha} \leq \bar{\sigma_1}}{\Gamma^* \vdash \langle e' e'' \rangle : [\alpha^*/\tau][\alpha'/\tau', \alpha''/\tau'']\sigma_2^*}$$

where $\bar{\alpha}$ and $\bar{\sigma_1}$ include three cases: $\bar{\alpha} = \alpha'$ and $\bar{\sigma_1} = \sigma_1'$, $\bar{\alpha} = \alpha''$ and $\bar{\sigma_1} = \sigma_1''$, and $\bar{\alpha} = \alpha^*$ and $\bar{\sigma_1} = \sigma_1^*$.

- If $\Gamma \vdash e : \sigma$ is got by using the rule Sub, then $\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$ is got from

$$\frac{\Gamma^* \vdash \langle e', e'' \rangle : \sigma^* \quad \hat{\Gamma}^* \vdash \sigma^* \leq \rho^*}{\Gamma^* \vdash \langle e', e'' \rangle : \rho^*}$$

Proposition 6.2 [Abstraction Theorem] *If $\Gamma \vdash e : \sigma$ is a valid typing judgement in $F_\wedge$, then*

$$\Gamma^* \vdash \langle e', e'' \rangle : \sigma^*$$

is valid in **Target**.

7 Conclusion

We presented a syntactic treatment of parametricity of the Girard-Reynolds system F in **Target**—a variant of Pierce's $F_\wedge$, and extended it to the study of parametricity of the system $F_\wedge$. We showed how parametricity conditions are represented as subtyping, and reasoning about parametricity becomes a simple matter of reasoning about the well-typedness of terms in a syntactic formalism. Some semantic issues are discussed and an initial result about parametricity of PER models is given. The work illustrates the expressiveness of the languages that combine intersection types and bounded polymorphism.

However, due to the fact that a general framework for models of bounded polymorphism is not yet available, we have not yet established the soundness of this approach with respect to the logical relations developed in [15]. Due to the same reason, we are not clear either how expressive the languages like $F_\wedge$ are. Though, with refer to Mairson's work [16], the internal logic of models of these languages should have the power of fragments of second-order logic systems that are needed for defining second-order logical relations.

The work in this paper is still in its early stage. To study binary relations, some specical subtyping, typing, and equality rules employed in **Target** are not yet well-understood. Though, this understanding may require a better understanding of subtyping and bounded quantification.

Another interesting work is to investigate whether our method really makes parametricity more accessible in functional programming and compiler construction.

One concern to our work is Pierce's recent work ([21]) on undecidability of bounded quantification. This immediately leads to the fact that subtyping in **Target** is undecidable as well. However, for the sake of parametricity, a weak form of All

$$\frac{\Theta, \tau \leq \sigma_1 \vdash \rho_2 \leq \sigma_2}{\Theta \vdash (\forall \tau \leq \sigma_1.\ \rho_2) \leq (\forall \tau \leq \sigma_1.\ \sigma_2)}$$

can be used in $F_\wedge$. Even with the replacement, it is still unknown whether subtyping in **Target** is decidable or not, since some special subtyping rules (e.g., Q-LB) are used.

Another concern to our work is Mairson's work in [16], where an attempt to a proof theory study of parametricity in second-order logic systems was made. For a particular application of parametricity, a particular semantic model is needed. But it is unclear how to get a model for second-order logic form a model of F. Practically, the problem with second-order logic is that the axioms for defining parametricity are complicated for the cases of higher functions and, therefore, difficult to use. To manage these formulas is obviously more difficult than to reason about the well-typedness in a programming environment.

Acknowledgements I wish to thank Luca Cardelli, Andrzej Filinski, Tim Freeman, Robert Harper, and Frank Pfenning for their encouragement and discussions. My advisor, John Reynolds, substantially improved a draft of this paper. Special thanks are due to Benjamin Pierce, with whom I had many illuminating discussions about $F_\wedge$ and the work of this paper.

References

[1] Samson Abramsky and Thomas P. Jensen. A relational approach to strictness analysis for higher-order polymorphic functions. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 49–54, 1991.

[2] Samson Abramsky, John C. Mitchell, Andre Scedrov, and Philip Wadler. Relators. Unpublished draft, 1991.

[3] E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70(1):35–64, January 15 1990. Corrigendum in (3) 71, 10 April 1990, p. 431.

[4] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–222, July 1991.

[5] Luca Cardelli. Typeful programming. Research Report 45, Digital Equipment Corporation, Systems Research Center, Palo Alto, California, February 1989.

[6] Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An extension of system F with subtyping. In *Theoretical Aspects of Computer Science*, 1991. To appear.

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

[8] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27:45–58, 1981.

[9] Pierre-Louis Curien and Giorgio Ghelli. Coherence of subsumption. *Mathematical Structures in Computer Science*, 1991. To appear.

[10] Pierre-Louis Curien and Giorgio Ghelli. Subtyping + extensionality: confluence of $\beta\eta$-reductions in $F_\leq$. In *Theoretical Aspects of Computer Science*, 1991. To appear.

[11] Peter J. Freyd. Structural polymorphism. Unpublished, January 23 1989.

[12] Giorgio Ghelli. *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* PhD thesis, Università di Pisa, March 1990. Technical report TD–6/90, Dipartimento di Informatica, Università di Pisa.

[13] Girard, J.-Y. Une extension de l'interpretation de Gödel à l'analyse, et son application a l'elimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92, Amsterdam, 1971. North-Holland.

[14] Gary T. Leavens and Don Pigozzi. Typed homomorphic relations extended with subtypes. In *Proceedings of Mathematical Foundations of Programming Semantics*, March 1991. To appear.

[15] Qingming Ma and John C. Reynolds. Types, abstraction, and parametric polymorphism: Part 2. In *Proceedings of Mathematical Foundations of Programming Semantics*, March 1991. To appear.

[16] Harry G. Mairson. Outline of a proof theory of parametricity. In *Proceedings of Fifth International Conference on Functional Programming Languages and Computer Architecture*, pages 313–327. Springer-Verlag, 1991.

[17] John C. Mitchell and Albert R. Meyer. Second-order logical relations. In R. Parikh, editor, *Proceedings of Conference on Logic of Programs*, Lecture Notes in Computer Science, pages 225–236, Berlin, 1985. Springer-Verlag.

[18] Frank Pfenning and Peter Lee. Metacircularity in the polymorphic lambda-calculus. *Theoretical Computer Science*, 1990. To appear. A preliminary version appeared in *TAPSOFT '89, Proceedings*

of the International Joint Conference on Theory and Practice in Software Development, Barcelona, Spain, pages 345–359, Springer-Verlag LNCS 352, March 1989.

[19] Frank Pfenning and Christine Paulin-Mohring. Inductively defined types in the Calculus of Constructions. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proceedings of the Fifth Conference on the Mathematical Foundations of Programming Semantics, Tulane University, New Orleans, Louisiana*, pages 209–228. Springer-Verlag LNCS 442, March 1989.

[20] Benjamin C. Pierce. Bounded quantification is undecidable. Technical Report CMU-CS-91-161, Carnegie Mellon University, July 1991.

[21] Benjamin C. Pierce. *Programming with Intersection Types and Bounded Polymorphism*. PhD thesis, Carnegie Mellon University, December 1991. To appear.

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

[23] John C. Reynolds. Towards a theory of type structure. In *Proceedings, Colloque sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425, Berlin, 1974. Springer-Verlag.

[24] John C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523, Amsterdam, 1983. Elsevier Science Publishers B. V. (North-Holland).

[25] John C. Reynolds. Preliminary design of the programming language forsythe. Report CMU-CS-88-159, Carnegie Mellon University, June 1988.

[26] John C. Reynolds. The coherence of languages with intersection types. In *Theoretical Aspects of Computer Science*, September 1991. To appear.

[27] Philip Wadler. Theorems for free. In *Proceedings of Fourth International Conference on Functional Programming Languages and Computer Architecture*, pages 347–359. ACM press, 1989.

Subtyping rules of $F_\wedge$

$$\Theta \vdash \sigma \leq \sigma \qquad \text{(Refl)}$$

$$\frac{\Theta \vdash \sigma_1 \leq \sigma_2 \quad \Theta \vdash \sigma_2 \leq \sigma_3}{\Theta \vdash \sigma_1 \leq \sigma_3} \qquad \text{(Trans)}$$

$$\frac{\tau \leq \sigma \in \Theta}{\Theta \vdash \tau \leq \sigma} \qquad \text{(Tvar)}$$

$$\frac{\Theta \vdash \sigma_1 \leq \rho_1 \quad \Theta \vdash \rho_2 \leq \sigma_2}{\Theta \vdash \rho_1 \to \rho_2 \leq \sigma_1 \to \sigma_2} \qquad \text{(Arrow)}$$

$$\frac{\Theta \vdash \sigma_1 \leq \rho_1 \quad \Theta, \tau \leq \sigma_1 \vdash \rho_2 \leq \sigma_2}{\Theta \vdash (\forall \tau \leq \rho_1.\ \rho_2) \leq (\forall \tau \leq \sigma_1.\ \sigma_2)} \qquad \text{(All)}$$

$$\frac{\Theta \vdash \rho \leq \sigma_1 \quad \Theta \vdash \rho \leq \sigma_2}{\Theta \vdash \rho \leq \sigma_1 \wedge \sigma_2} \qquad \text{(Inter-G)}$$

$$\Theta \vdash \sigma_1 \wedge \sigma_2 \leq \sigma_i \ \text{ for } \ i = 1,2 \qquad \text{(Inter-LB)}$$

$$\Theta \vdash (\rho \to \sigma_1) \wedge (\rho \to \sigma_2) \leq \rho \to (\sigma_1 \wedge \sigma_2) \qquad \text{(Dist-IA)}$$

$$\Theta \vdash (\forall \tau \leq \sigma.\sigma_1) \wedge (\forall \tau \leq \sigma.\sigma_2) \leq (\forall \tau \leq \sigma.\ \sigma_1 \wedge \sigma_2) \qquad \text{(Dist-IQ)}$$

$$\Theta \vdash \rho \leq \mathbf{ns} \qquad \text{(Ns-G)}$$

$$\Theta \vdash \mathbf{ns} \leq \rho \to \mathbf{ns} \qquad \text{(Ns-IA)}$$

$$\Theta \vdash \mathbf{ns} \leq \forall \tau \leq \sigma.\ \mathbf{ns} \qquad \text{(Ns-IQ)}$$

$$\Theta \vdash \mathbf{ns} \leq \mathbf{ns} \times \mathbf{ns} \qquad \text{(Ns-IP)}$$

Typing rules of $F_\wedge$

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \text{(Var)}$$

$$\frac{\Gamma; x : \sigma_1 \vdash e : \sigma_2}{\Gamma \vdash \lambda x.\ e : \sigma_1 \to \sigma_2} \qquad \text{(Arrow-I)}$$

$$\frac{\Gamma \vdash e_2 : \sigma_1 \to \sigma_2 \quad \Gamma \vdash e_1 : \sigma_1}{\Gamma \vdash e_2 e_1 : \sigma_2} \qquad \text{(Arrow-E)}$$

$$\frac{\Gamma \vdash e : \sigma_1 \quad \Gamma \vdash e : \sigma_2}{\Gamma \vdash e : \sigma_1 \wedge \sigma_2} \qquad \text{(Inter-I)}$$

$$\frac{\Gamma; \tau \leq \sigma_1 \vdash e : \sigma_2}{\Gamma \vdash e : \forall \tau \leq \sigma_1.\ \sigma_2} \qquad \text{(All-I)}$$

$$\frac{\Gamma \vdash e : \forall \tau \leq \sigma_1.\ \sigma_2 \quad \hat{\Gamma} \vdash \sigma \leq \sigma_1}{\Gamma \vdash e : [\sigma/\tau]\sigma_2} \qquad \text{(All-E)}$$

$$\Gamma \vdash e : \mathbf{ns} \qquad \text{(Ns-I)}$$

$$\frac{\Gamma \vdash e : \sigma_1 \quad \hat{\Gamma} \vdash \sigma_1 \leq \sigma_2}{\Gamma \vdash e : \sigma_2} \qquad \text{(Sub)}$$

Equational theory of $F_\wedge$

$$\frac{\Gamma \vdash e = e' : \sigma}{\Gamma \vdash e' = e' : \sigma} \qquad \text{(Symm)}$$

$$\frac{\Gamma \vdash e = e' : \sigma \qquad \Gamma \vdash e' = e'' : \sigma}{\Gamma \vdash e = e'' : \sigma} \quad (\textsc{Tran})$$

$$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash e = e : \sigma} \quad (\textsc{Refl-E})$$

$$\Gamma \vdash e = e' : \mathbf{ns} \quad (\textsc{Ns})$$

$$\frac{\Gamma \vdash e = e' : \sigma_1 \qquad \Gamma \vdash e = e' : \sigma_2}{\Gamma \vdash e = e' : \sigma_1 \wedge \sigma_2} \quad (\textsc{Inte})$$

$$\frac{\Gamma; x : \sigma_1 \vdash e = e' : \sigma_2}{\Gamma \vdash \lambda x.\ e = \lambda x.\ e' : \sigma_1 \rightarrow \sigma_2} \quad (\textsc{Abs-Term})$$

$$\frac{\Gamma; \tau \leq \sigma_1 \vdash e = e' : \sigma_2}{\Gamma \vdash e = e' : \forall \tau \leq \sigma_1.\ \sigma_2} \quad (\textsc{Abs-Type})$$

$$\frac{\Gamma \vdash e_1 = e_1' : \sigma_1 \quad \Gamma \vdash e_2 = e_2' : \sigma_1 \rightarrow \sigma_2}{\Gamma \vdash e_2 e_1 = e_2' e_1' : \sigma_2} \quad (\textsc{Appl-Term})$$

$$\frac{\Gamma \vdash e = e' : \forall \tau \leq \sigma_1.\ \sigma_2 \qquad \hat{\Gamma} \vdash \alpha \leq \sigma_1}{\Gamma \vdash e = e' : [\alpha/\tau]\sigma_2} \quad (\textsc{Appl-Type})$$

$$\frac{\Gamma \vdash e_1 = e_1' : \sigma_1 \qquad \Gamma; x : \sigma_1 \vdash e_2 = e_2' : \sigma_2}{\Gamma \vdash (\lambda x.\ e_2) e_1 = [e_1'/x] e_2' : \sigma_2} \quad (\textsc{Beta})$$

$$\frac{\Gamma \vdash e = e' : \sigma_1 \rightarrow \sigma_2 \qquad y \notin \mathrm{dom}\,\Gamma}{\Gamma \vdash \lambda y.\ e y = e' : \sigma_1 \rightarrow \sigma_2} \quad (\textsc{Eta})$$

Subtyping rules of Target that are not in $F_\wedge$

$$\frac{\vdash \sigma \leq_{\mathrm{pre}} \sigma'}{\Theta \vdash \sigma \leq \sigma'} \quad (\textsc{Pre-Def})$$

$$\frac{\Theta \vdash \rho \leq \sigma_1}{\Theta \vdash (\forall \tau \leq \sigma_1.\ \sigma_2) \leq [\rho/\tau]\sigma_2} \quad (\text{Q-LB})$$

$$\frac{\Theta \vdash \rho_1 \leq \sigma_1 \quad \Theta \vdash \rho_2 \leq \sigma_2}{\Theta \vdash \rho_1 \times \rho_2 \leq \sigma_1 \times \sigma_2} \quad (\textsc{Prod})$$

$$\Theta \vdash (\sigma_1 \rightarrow \rho_1) \times (\sigma_2 \rightarrow \rho_2) \leq (\sigma_1 \wedge \sigma_2) \rightarrow (\rho_1 \wedge \rho_2) \quad (\textsc{Dist-PA})$$

$$\Theta \vdash (\rho_1 \times \rho_2) \wedge (\sigma_1 \times \sigma_2) \leq (\rho_1 \wedge \sigma_1) \times (\rho_2 \wedge \sigma_2) \quad (\textsc{Dist-IP})$$

$$\Theta \vdash (\forall \tau_1 \leq \rho_1 \forall \tau_2 \leq \rho_2.\ \sigma_1 \times \sigma_2) \leq \theta_1 \times \theta_2 \quad (\textsc{Dist-QP})$$

$[\theta_i \equiv (\forall \tau_i \leq \rho_i.\ \sigma_i),\ \tau_1 \notin \sigma_2,\ \text{and}\ \tau_2 \notin \sigma_1]$

Typing rules of Target that are not in $F_\wedge$

$$\frac{\Gamma \vdash e_1 : \sigma_1 \quad \Gamma \vdash e_2 : \sigma_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \sigma_1 \times \sigma_2} \quad (\textsc{Prod-I})$$

$$\frac{\Gamma \vdash e : \sigma_1 \times \sigma_2}{\Gamma \vdash \mathrm{fst}(e) : \sigma_1} \qquad \frac{\Gamma \vdash e : \sigma_1 \times \sigma_2}{\Gamma \vdash \mathrm{snd}(e) : \sigma_2} \quad (\textsc{Prod-E})$$

$$\frac{\Gamma; \langle x', x'' \rangle : \sigma_1 \vdash \langle e',\ e'' \rangle : \sigma_2 \quad x' \notin e''\ \ x'' \notin e'}{\Gamma \vdash \langle \lambda x'.\ e',\ \lambda x''.\ e'' \rangle : \sigma_1 \rightarrow \sigma_2} \quad (\textsc{Arrow-I'})$$

$$\frac{\Gamma \vdash \langle e_1',\ e_1'' \rangle : \sigma_1 \rightarrow \sigma_2 \quad \Gamma \vdash \langle e_2',\ e_2'' \rangle : \sigma_1}{\Gamma \vdash \langle e_1' e_2',\ e_2' e_2'' \rangle : \sigma_2} \quad (\textsc{Arrow-E'})$$

Equality rules of Target that are not in $F_\wedge$

$$\frac{\Gamma \vdash e_1 = e_1' : \sigma_1 \quad \Gamma \vdash e_2 = e_2' : \sigma_2}{\Gamma \vdash \langle e_1,\ e_2 \rangle = \langle e_1',\ e_2' \rangle : \sigma_1 \times \sigma_2} \quad (\textsc{Pair})$$

$$\frac{\Gamma \vdash e = e' : \sigma_1 \times \sigma_2}{\Gamma \vdash \mathrm{fst}(e) = \mathrm{fst}(e') : \sigma_1} \quad \frac{\Gamma \vdash e = e' : \sigma_1 \times \sigma_2}{\Gamma \vdash \mathrm{snd}(e) = \mathrm{snd}(e') : \sigma_2} \quad (\textsc{Proj})$$

$$\frac{\Gamma \vdash \langle e_1, e_2 \rangle = \langle e_1', e_2' \rangle : \sigma_1 \times \sigma_2}{\Gamma \vdash \mathrm{fst} \langle e_1, e_2 \rangle = e_1' : \sigma_1} \quad (\textsc{Fst})$$

$$\frac{\Gamma \vdash \langle e_1, e_2 \rangle = \langle e_1', e_2' \rangle : \sigma_1 \times \sigma_2}{\Gamma \vdash \mathrm{snd} \langle e_1, e_2 \rangle = e_2' : \sigma_2} \quad (\textsc{Snd})$$

$$\frac{\Gamma; \langle x_1, x_2 \rangle : \sigma_1 \vdash \langle e_1, e_2 \rangle = \langle e_1', e_2' \rangle : \sigma_2}{\Gamma \vdash \langle \lambda x_1.e_1, \lambda x_2.e_2 \rangle = \langle \lambda x_1.e_1', \lambda x_2.e_2' \rangle : \sigma_1 \rightarrow \sigma_2} \quad (\textsc{Abs-Terms'})$$

$[x_1 \notin e_2,\ x_1 \notin e_2',\ x_2 \notin e_1,\ x_2 \notin e_1']$

$$\frac{\Gamma \vdash \langle d_1, d_2 \rangle = \langle d_1', d_2' \rangle : \sigma_1 \quad \Gamma \vdash \langle e_1, e_2 \rangle = \langle e_1', e_2' \rangle : \sigma_1 \rightarrow \sigma_2}{\Gamma \vdash \langle e_1 d_1, e_2 d_2 \rangle = \langle e_1' d_1', e_2' d_2' \rangle : \sigma_2} \quad (\textsc{Appl-Terms'})$$

$$\frac{\Gamma \vdash \langle d_1, d_2 \rangle = \langle d_1', d_2' \rangle : \sigma_1 \quad \Gamma; \langle x_1, x_2 \rangle : \sigma_1 \vdash \langle e_1, e_2 \rangle = \langle e_1', e_2' \rangle : \sigma_2}{\Gamma \vdash \langle a_1, a_2 \rangle = \langle [d_1'/x_1] e_1', [d_2'/x_2] e_2' \rangle : \sigma_2} \quad (\textsc{Beta'})$$

$[a_i \equiv (\lambda x_i.e_i) d_i,\ x_1 \notin e_2,\ x_1 \notin e_2',\ x_2 \notin e_1,\ x_1 \notin e_1']$

$$\frac{\Gamma \vdash \langle e_1, e_2 \rangle = \langle e_1', e_2' \rangle : \sigma_1 \rightarrow \sigma_2 \qquad y \notin \mathrm{dom}\,\Gamma}{\Gamma \vdash \langle \lambda y.e_1 y, \lambda y.e_2 y \rangle = \langle e_1', e_2' \rangle : \sigma_1 \rightarrow \sigma_2} \quad (\textsc{Eta'})$$