Advances in Type Systems for Functional Logic Programming (Extended Version)*

Technical Report SIC-05-12

Francisco J. López-Fraguas
Enrique Martin-Martin
Juan Rodríguez-Hortalá

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
fraguas@sip.ucm.es, emartinm@fdi.ucm.es, juanrh@fdi.ucm.es

Abstract. Type systems are widely used in programming languages as a powerful tool providing safety to programs, and forcing the programmers to write code in a clearer way. Functional-logic languages have inherited Damas & Milner type system from their functional part due to its simplicity and popularity. In this paper we address a couple of aspects that can be subject of improvement. One is related to a problematic feature of functional logic languages not taken under consideration by standard systems: it is known that the use of *opaque* HO patterns in left-hand sides of program rules may produce undesirable effects from the point of view of types. We re-examine the problem, and propose a Damas & Milner-like type system where certain uses of HO patterns (even opaque) are permitted while preserving type safety, as proved by a subject reduction result that uses *HO-let-rewriting*, a recently proposed operational semantics for HO functional logic programs. At the same time that we formalize the type system, we have made the effort of technically clarifying additional issues: one is the different ways in which polymorphism of local definitions can be handled, and the other is the overall process of type inference in a whole program.

1 Introduction

Type systems for programming languages are an active area of research [17], no matters which paradigm one considers. In the case of functional programming, most type systems have arisen as extensions of Damas & Milner's [3], for its remarkable simplicity and good properties (decidability, existence of principal types, possibility of type inference). Functional logic languages [11,7,6], in their practical side, have inherited more or less directly Damas & Milner's types.

* This paper is the extended version of "**Advances in Type Systems for Functional Logic Programming**" appeared in *Pre-proceedings of the 18th International Workshop on Functional and (Constraint) Logic Programming (WFLP'09)*, June 28, 2009, Brasília, Brazil.

In principle, most of the type extensions proposed for functional programming could be also incorporated to functional logic languages (this has been done, for instance, for type classes in [14]). However, if types are not only decoration but are to provide safety, one should be sure that the adopted system has indeed good properties. In this paper we tackle a couple of aspects of existing FLP systems that are problematic or not well covered by standard Damas & Milner systems. One is the presence of so called *HO patterns* in programs, an expressive feature for which a sensible semantics exists [4]; however, it is known that unrestricted use of HO patterns leads to type unsafety, as recalled below. The second is the degree of polymorphism assumed for local pattern bindings, a matter with respect to which existing FP or FLP systems vary greatly.

The rest of the paper is organized as follows. The next two subsections further discuss the two mentioned aspects. Section 2 contains some preliminaries about FL programs and types. In Section 3 we expose the type system and prove its soundness wrt. the *let rewriting* semantics of [10]. Section 4 contains a type inference relation, which let us find the most general type of expressions. Section 5 present a method to infer types for programs. Finally, Section 6 contains some conclusions and future work.

1.1 Higher order patterns

In our formalism patterns appear in the left-hand side of rules and in lambda or let expressions. Some of these patterns can be HO patterns, if they contain partial applications of function or constructor symbols. HO patterns can be a source of problems from the point of view of the types. In particular, it was shown in [5] that unrestricted use of HO patterns leads to loss of expected property of *subject reduction* (i.e., evaluation does not change types), an essential property for a type system. The following is a crisp example of the problem.

Example 1 (Polymorphic Casting [2]). Consider the program consisting of the rules $snd\ X\ Y \to Y$, $and\ true\ X \to X$, $and\ false\ X \to false$, $id\ X \to X$, with the usual types inferred by a classical Damas & Milner algorithm. Then we can write the functions $co\ (snd\ X) \to X$ and $cast\ X \to co\ (snd\ X)$, whose inferred types will be $\forall\alpha.\forall\beta.(\alpha \to \alpha) \to \beta$ and $\forall\alpha.\forall\beta.\alpha \to \beta$ respectively. It is clear that $and\ (cast\ 0)\ true$ is well-typed, because $cast\ 0$ has type $bool$ (in fact it has any type), but if we reduce the expression to $and\ 0\ true$ using the rule of $cast$ the resulting expression is bad-typed.

The problem arises when dealing with HO patterns, because unlike FO patterns, knowing the type of a pattern does not always permit us to know the type of its subpatterns. In the previous example the cause is function co, because its pattern $snd\ X$ is *opaque* and shadows the type of its subpattern X. Usual inference algorithms treat this opacity as polymorphism, and that is the reason why it is inferred a completely polymorphic type for the the result of the function co.

In [5] the appearance of any opaque pattern in the left-hand side of the rules is prohibited, but we will see that it is possible to be less restrictive. The key is

making a distinction between **opaque** and **transparent** variables of a pattern: a variable is opaque if its type is not univocally fixed by the type of the pattern, and is transparent otherwise. We call a variable of a pattern **critical** if it is opaque in the pattern and also appears elsewhere in the expression. The formal definition of opaque and critical variables will be given in Sect. 3. With these notions we can relax the situation in [5], prohibiting only those patterns having critical variables.

1.2 Local definitions

Functional and functional logic languages provide syntax to introduce local definitions inside an expression. But in spite of the popularity of let-expressions, different implementations treat them differently because of the polymorphism they give to bound variables. This differences can be observed in Example 2, being $(e_1, \ldots, e_n)$ and $[e_1, \ldots, e_n]$ the usual tuple and list notation respectively.

Example 2 (let expressions). Let e_1 be $let\ F = id\ in\ (F\ true, F\ 0)$, and e_2 be $let\ [F, G] = [id, id]\ in\ (F\ true, F\ 0, G\ 0, G\ false)$

Intuitively, e_1 gives a new name to the identity function and uses it twice with arguments of different types. Surprisingly, not all the implementations consider this expression as well-typed, and the reason is that F is used with different types in each appearance: $bool \to bool$ and $int \to int$. Some implementations as Clean 2.2, PAKCS 1.9.1 or KICS 0.81893 consider that a variable bound by a let-expression must be used with the same type in all the appearances in the body of the expression. In this situation we say that lets are completely monomorphic, and write let_m for it.

On the other hand, we can consider that all the variables bound by the let-expression may have different but coherent types, i.e., are treated polymorphically. Then expressions like e_1 or e_2 would be well-typed. This is the decision adopted by Hugs Sept 2006 or OCaml 3.10.2. In this case, we will say that lets are completely polymorphic, and write let_p.

Finally, we can treat the bound variables monomorphically or polymorphically depending on the form of the pattern. If the pattern is a variable, the let treats it polymorphically, but if it is compound the let treats all the variables monomorphically. This is the case of GHC 6.8.2, SML of New Jersey v110.67 or Curry Münster 0.9.11. In this implementations e_1 is well-typed, while e_2 not. We call this kind of let-expression let_{pm}.

Fig. 1 summarizes the decisions of various implementations of functional and functional logic languages. The exact behavior wrt. types of local definitions is usually not well documented, not to say formalized, in those system. One of our contributions is this paper is to technically clarify this question by adopting a neutral position, and formalizing the different possibilities for the polymorphism of local definitions.

Programming language and version	let_m	let_{pm}	let_p
GHC 6.8.2		×	
Hugs Sept. 2006			×
Standard ML of New Jersey 110.67		×	
Ocaml 3.10.2			×
Clean 2.0	×		
$\mathcal{TOY}$ 2.3.1*	×		
Curry PAKCS 1.9.1	×		
Curry Münster 0.9.11		×	
KICS 0.81893	×		

(*) we use `where` instead of `let`, not supported by $\mathcal{TOY}$

Fig. 1. Let expressions in different programming languages.

2 Preliminaries

We assume a signature $\Sigma = DC \cup FS$, where DC and FS are two disjoint set of *data constructor* and *function* symbols resp., all them with associated arity. We write DC^n (resp FS^n) for the set of constructor (function) symbols of arity n. We also assume a numerable set $\mathcal{DV}$ of *data variables* X. We define the set of *patterns* $Pat \ni t ::= X \mid f \mid c\ t_1 \dots t_n\ (n \leq m) \mid f\ t_1 \dots t_n (n < m)$ and the set of *expressions* $Exp \ni e ::= X \mid c \mid f \mid e_1\ e_2 \mid \lambda t.e \mid let_m\ t = e_1\ in\ e_2 \mid let_{pm}\ t = e_1\ in\ e_2 \mid let_p\ t = e_1\ in\ e_2$ where $c \in DC^m$ and $f \in FS^m$. We split the set of patterns in two: *first order patterns* $FOPat \ni fot ::= X \mid c\ t_1 \dots c_n$ where $c \in DC^n$, and *Higher order patterns* $HOPat = Pat \setminus FOPat$. Expressions $h\ e_1 \dots e_m$ are called *junk* if $h \in CS^n$ and $m > n$, and *active* if $h \in FS^n$ and $m \geq n$. $FV(e)$ is the set of variables in e which are not bound by any lambda or let expression and is defined in the usual way (notice that since our let expressions do not support recursive definitions the bindings of the pattern only affect e_2: $FV(let_*\ t = e_1\ in\ e_2) = FV(e_1) \cup (FV(e_2) \setminus var(t))$. A *one-hole context* $\mathcal{C}$ is an expression with exactly one hole. A *data substitution* $\theta \in \mathcal{PS}ubst$ is a finite mapping from data variables to patterns: $[\overline{X_i/t_i}]$. Substitution application over data variables and expressions is defined in the usual way. A *program rule* is defined as $PRule \ni r ::= f\ t_1 \dots t_n \to e\ (n \geq 0)$ where the set of patterns $\overline{t_i}$ is linear and $FV(e) \subseteq \bigcup_i FV(t_i)$. Therefore, extra variables are not considered in this paper. A program is a set of program rules $Prog \ni \mathcal{P} ::= \{r_1; \dots; r_n\} (n \geq 0)$.

For the types we assume a numerable set $\mathcal{TV}$ of *type variables* α and a countable alphabet $\mathcal{TC} = \bigcup_{n \in \mathbb{N}} \mathcal{TC}^n$ of *type constructors* C. The set of *simple types* is defined as $SType \ni \tau ::= \alpha \mid \tau_1 \to \tau_2 \mid C\ \tau_1 \dots \tau_n\ (C \in \mathcal{TC}^n)$. Based on simple types we can define the set of *type-schemes* as $TScheme \ni \sigma ::= \tau \mid \forall \alpha.\sigma$. The set of *free type variables* (FTV) of a simple type τ is $var(\tau)$, and for type-schemes $FTV(\forall \overline{\alpha_i}.\tau) = FTV(\tau) \setminus \{\overline{\alpha_i}\}$. A type-scheme $\forall \overline{\alpha_i}.\overline{\tau_n} \to \tau$ is *transparent* if $FTV(\overline{\tau_n}) \subseteq FTV(\tau)$. A *set of assumptions* $\mathcal{A}$ is $\{\overline{s_i : \sigma_i}\}$, where $s_i \in DC \cup FS \cup \mathcal{DV}$. Notice that the transparency of type-schemes for data constructors is not required in our setting, although that hypothesis is usually assumed in classical Damas & Milner type systems. If $(s_i : \sigma_i) \in \mathcal{A}$

we write $\mathcal{A}(s_i) = \sigma_i$. A *type substitution* $\pi \in \mathcal{TS}ubst$ is a finite mapping from type variables to simple types $[\overline{\alpha_i/\tau_i}]$. For sets of assumptions $FTV(\{\overline{s_i : \sigma_i}\}) = \bigcup_i FTV(\sigma_i)$. We will say a type-scheme σ is *closed* if $FTV(\sigma) = \emptyset$. The notion of applying a type substitution to a type variable or simple type is the natural, and for type-schemes consists in applying the substitution only to their free variables. This notion is extended to set of assumptions in the obvious way. We will say σ is an *instance* of σ' if $\sigma = \sigma'\pi$ for some π. τ' is a *generic instance* of $\sigma \equiv \forall\overline{\alpha_i}.\tau$ if $\tau' = \tau[\overline{\alpha_i/\tau_i}]$ for some $\overline{\tau_i}$, and we write it $\sigma \succ \tau'$. We extend $\succ$ to a relation between type-schemes by saying that $\sigma \succ \sigma'$ iff every simple type such that is a generic instance of σ' is also a generic instance of σ. Then $\forall\overline{\alpha_i}.\tau \succ \forall\overline{\beta_i}.\tau[\overline{\alpha_i/\tau_i}]$ iff $\{\overline{\beta_i}\} \cap FTV(\forall\overline{\alpha_i}.\tau) = \emptyset$ [12]. Finally, τ' is a *variant* of $\sigma \equiv \forall\overline{\alpha_i}.\tau$ ($\sigma \succ_{var} \tau'$) if $\tau' = \tau[\overline{\alpha_i/\beta_i}]$ and $\overline{\beta_i}$ are fresh type variables.

3 Type derivation

We propose a modification of Damas & Milner type system [3] with some differences. We have found convenient to separate the task of giving a regular Damas & Milner type and the task of checking critical variables. To do that we have defined two different type relations: $\vdash$ and $\vdash^{\bullet}$.

The basic typing relation $\vdash$ in the upper part of Fig. 2 is like the classical Damas & Milner's system but extended to handle the three different kinds of let expressions and the occurrence of patterns instead of variables in lambda and let expressions. We have also made the rules more syntax-directed so that the form of type derivations depends only on the form of the expression to be typed. $Gen(\tau, \mathcal{A})$ is the clausure or generalization of τ wrt. $\mathcal{A}$ [3,12,18], which generalizes all the type variables of τ that do not appear free in $\mathcal{A}$. Formally: $Gen(\tau, \mathcal{A}) = \forall\overline{\alpha_i}.\tau$ *where* $\{\overline{\alpha_i}\} = FTV(\tau) \setminus FTV(\mathcal{A})$. As can be seen, $[\mathrm{LET}_m]$ and $[\mathrm{LET}^h_{pm}]$ behave the same, and do not generalize any of the types τ_i for the variables X_i to give a type for the body. On the contrary, $[\mathrm{LET}^X_{pm}]$ and $[\mathrm{LET}_p]$ generalize the types given to the variables. Notice that if two variables share the same type in the set of assumptions $\mathcal{A}$, generalization will lose the connection between them. This fact can be seen with e_2 in Ex. 2. Although the type for both F and G can be $\alpha \to \alpha$ (with α a variable not appearing in $\mathcal{A}$) the generalization step will assign both the type-scheme $\forall\alpha.\alpha \to \alpha$, losing the connection between them.

The $\vdash^{\bullet}$ relation (lower part of Fig. 2) uses $\vdash$ but enforces also the absence of critical variables. The characterization of an *opaque variable* is defined as follows. It states that a variable X_i is opaque in t when it is possible to build a type derivation for t where the type assumed for X_i contains type variables which do not occur in the type derived for the pattern.

Definition 1 (Opaque variable of t wrt. $\mathcal{A}$). *Let t be a pattern that admits type wrt. a given set of assumptions $\mathcal{A}$. We say that $X_i \in \overline{X_i} = var(t)$ is opaque wrt. $\mathcal{A}$ iff $\exists\overline{\tau_i}, \tau$ s.t. $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau$ and $FTV(\tau_i) \not\subseteq FTV(\tau)$.*

The previous definition is based on the existence of a certain type derivation, and therefore cannot be used as an effective check for the opacity of variables. An equivalent characterization can be formulated exploiting the close relationship between $\vdash$ an type inference $\Vdash$ that will be presented in Sect. 4. Since $\Vdash$ can be viewed as an algorithm, Prop. 1 provides a more operational definition which is useful when implementing the type system.

[ID]	$\dfrac{}{\mathcal{A} \vdash s : \tau}$	if $\begin{array}{l} s \in DC \cup FS \cup \mathcal{DV} \\ \wedge\ (s : \sigma) \in \mathcal{A} \wedge\ \ \sigma \succ \tau \end{array}$
[APP]	$\dfrac{\mathcal{A} \vdash e_1 : \tau_1 \to \tau \quad \mathcal{A} \vdash e_2 : \tau_1}{\mathcal{A} \vdash e_1 e_2 : \tau}$	
$[\Lambda]$	$\dfrac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t \quad \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau}{\mathcal{A} \vdash \lambda t.e : \tau_t \to \tau}$	if $\{\overline{X_i}\} = var(t)$
$[\mathbf{LET}_m]$	$\dfrac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t \quad \mathcal{A} \vdash e_1 : \tau_t \quad \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e_2 : \tau_2}{\mathcal{A} \vdash \mathbf{let}_m\ t = e_1\ \mathbf{in}\ e_2 : \tau_2}$	if $\{\overline{X_i}\} = var(t)$
$[\mathbf{LET}^X_{pm}]$	$\dfrac{\mathcal{A} \vdash e_1 : \tau_1 \quad \mathcal{A} \oplus \{X : Gen(\tau_1, \mathcal{A})\} \vdash e_2 : \tau_2}{\mathcal{A} \vdash \mathbf{let}_{pm}\ X = e_1\ \mathbf{in}\ e_2 : \tau_2}$	
$[\mathbf{LET}^h_{pm}]$	$\dfrac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash h\ t_1 \ldots t_n : \tau_t \quad \mathcal{A} \vdash e_1 : \tau_t \quad \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e_2 : \tau_2}{\mathcal{A} \vdash \mathbf{let}_{pm}\ h\ t_1 \ldots t_n = e_1\ \mathbf{in}\ e_2 : \tau_2}$	if $\{\overline{X_i}\} = var(t_1 \ldots t_n)$
$[\mathbf{LET}_p]$	$\dfrac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t \quad \mathcal{A} \vdash e_1 : \tau_t \quad \mathcal{A} \oplus \{\overline{X_i : Gen(\tau_i, \mathcal{A})}\} \vdash e_2 : \tau_2}{\mathcal{A} \vdash \mathbf{let}_p\ t = e_1\ \mathbf{in}\ e_2 : \tau_2}$	if $\{\overline{X_i}\} = var(t)$
[P]	$\dfrac{\mathcal{A} \vdash e : \tau}{\mathcal{A} \vdash^{\bullet} e : \tau}$	if $critVar_{\mathcal{A}}(e) = \emptyset$

Fig. 2. Rules of type system

Proposition 1. *$X_i \in \overline{X_i} = var(t)$ is opaque wrt. $\mathcal{A}$ iff $\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_g | \pi_g$ and $FTV(\alpha_i \pi_g) \nsubseteq FTV(\tau_g)$.*

We write $opaqueVar_{\mathcal{A}}(t)$ for set of opaque variables of t wrt. $\mathcal{A}$. Now, we can define the *critical variables* of an expression e wrt. $\mathcal{A}$ as those variables that, being opaque in a let or lambda pattern of e, are indeed used in e. Formally:

Definition 2 (Critical variables).

$$critVar_{\mathcal{A}}(s) = \emptyset \qquad critVar_{\mathcal{A}}(e_1 e_2) = critVar_{\mathcal{A}}(e_1) \cup critVar_{\mathcal{A}}(e_2)$$
$$critVar_{\mathcal{A}}(\lambda t.e) = (opaqueVar_{\mathcal{A}}(t) \cap FV(e)) \cup critVar_{\mathcal{A}}(e)$$
$$critVar_{\mathcal{A}}(let_*\ t = e_1\ in\ e_2)$$
$$= (opaqueVar_{\mathcal{A}}(t) \cap FV(e_2)) \cup\ critVar_{\mathcal{A}}(e_1) \cup critVar_{\mathcal{A}}(e_2)$$

The typing relation $\vdash^\bullet$ has been defined in a modular way in the sense that the opacity check is kept separated from the regular Damas & Milner typing. Therefore it is easy to see that if every constructor and function symbol in program has a transparent assumption, then all the variables in patterns will be transparent, and so $\vdash^\bullet$ will be equivalent to $\vdash$. This happens in particular for those programs using only first order patterns and whose constructor symbols come from a Haskell (or Toy, Curry)-like `data` declaration.

3.1 Properties of the typing relations

The typing relations fulfill a set of useful properties. Here we use $\vdash^?$ for any of the two typing relations: $\vdash$ or $\vdash^\bullet$.

Theorem 1 (Properties of the typing relations).

a) *If* $\mathcal{A} \vdash^? e : \tau$ *then* $\mathcal{A}\pi \vdash^? e : \tau\pi$

b) *Let* s *be a symbol which does not appear in* e. *Then* $\mathcal{A} \vdash^? e : \tau \iff \mathcal{A} \oplus \{s : \sigma_s\} \vdash^? e : \tau$.

c) *If* $\mathcal{A} \oplus \{X : \tau_x\} \vdash^? e : \tau$ *and* $\mathcal{A} \oplus \{X : \tau_x\} \vdash^? e' : \tau_x$ *then* $\mathcal{A} \oplus \{X : \tau_x\} \vdash^? e[X/e'] : \tau$.

d) *If* $\mathcal{A} \oplus \{s : \sigma\} \vdash e : \tau$ *and* $\sigma' \succ \sigma$, *then* $\mathcal{A} \oplus \{s : \sigma'\} \vdash e : \tau$.

Part *a)* states that type derivations are closed under type substitutions. *b)* shows that type derivations for e depend only on the assumptions for the symbols in e. *c)* is a substitution lemma stating that in a type derivation we can replace a variable by an expression with the same type. Finally, *d)* establishes that from a valid type derivation we can change the assumption of a symbol for a more general type-scheme, and we still have a correct type derivation for the same type. Notice that this is not true wrt. the typing relation $\vdash^\bullet$ because a more general type can introduce opacity. For example the variable X is opaque in *snd* X with the usual type for *snd*, but with a more specific type such as $bool \to bool \to bool$ it is no longer opaque.

3.2 Subject Reduction

Subject reduction is a key property for type systems, meaning that evaluation does not change the type of an expression. This ensures that run-time type errors will not occur. Subject reduction is only guaranteed for *well-typed* programs, a notion that we formally define now.

Definition 3 (Well-typed program). *A program rule* $f\ t_1 \ldots t_n \to e$ *is well-typed wrt.* $\mathcal{A}$ *if* $\mathcal{A} \vdash^{\bullet} \lambda t_1 \ldots \lambda t_n.e : \tau$ *and* τ *is a variant of* $\mathcal{A}(f)$*. A program* $\mathcal{P}$ *is well-typed wrt.* $\mathcal{A}$ *if all its rules are well-typed wrt.* $\mathcal{A}$*. If* $\mathcal{P}$ *is well-typed wrt.* $\mathcal{A}$ *we write* $wt_{\mathcal{A}}(\mathcal{P})$*.*

Notice the use of the extended typing relation $\vdash^{\bullet}$ in the previous definition. This is essential, as we will explain later.

Although the restriction that the type of the lambda abstraction associated to a rule must be a variant of the type of the function symbol (and not an instance) may be strange, it is necessary. If not, the fact that a program is well-typed will not give us important information about the functions like the type of their arguments, and will make us to consider as well-typed undesirable programs like $\mathcal{P} \equiv \{f\ true \to true; f\ 2 \to false\}$ with the assumptions $\mathcal{A} \equiv \{f :: \forall\alpha.\alpha \to bool\}$. Besides, this restriction is implicitly considered in [5].

$$
\begin{array}{rl}
TRL(s) = s, & \text{if } s \in DC \cup FS \cup \mathcal{DV} \\
TRL(e_1\ e_2) = & TRL(e_1)\ TRL(e_2) \\
TRL(let_K\ X = e_1\ in\ e_2) = & let_K\ X = TRL(e_1)\ in\ TRL(e_2), \text{ with } K \in \{m, p\} \\
TRL(let_{pm}\ X = e_1\ in\ e_2) = & let_p\ X = TRL(e_1)\ in\ TRL(e_2) \\
TRL(let_m\ t = e_1\ in\ e_2) = & let_m\ Y = TRL(e_1)\ in\ \overline{let_m\ X_i = f_{X_i}\ Y}\ in\ TRL(e_2) \\
TRL(let_{pm}\ t = e_1\ in\ e_2) = & let_m\ Y = TRL(e_1)\ in\ \overline{let_m\ X_i = f_{X_i}\ Y}\ in\ TRL(e_2) \\
TRL(let_p\ t = e_1\ in\ e_2) = & let_p\ Y = TRL(e_1)\ in\ \overline{let_p\ X_i = f_{X_i}\ Y}\ in\ TRL(e_2)
\end{array}
$$

for $\{\overline{X_i}\} = var(t) \cap var(e_2)$, $f_{X_i} \in FS^1$ fresh defined by the rule $f_{X_i}\ t \to X_i$, $Y \in \mathcal{DV}$ fresh, t a non variable pattern and t' any pattern.

Fig. 3. Transformation rules of let expressions with patterns

For subject reduction to be meaningful, a notion of evaluation is needed. In this paper we consider the *let-rewriting* relation of [10]. As can be seen, *let-rewriting* does not support let expressions with compound patterns. Instead of extending the semantics with this feature we propose a transformation from let-expressions with patterns to let-expressions with only variables (Fig. 3). There are various ways to perform this transformation, which differ in the strictness of the pattern matching. We have chosen the alternative explained in [16] that does not demand the matching if no variable of the pattern is needed, but otherwise forces the matching of the whole pattern. This transformation has been enriched with the different kinds of let expressions in order to preserve the types, as is stated in Th. 2. Notice that the result of the transformation and the expressions accepted by *let-rewriting* only has let_m or let_p expressions, since without compound patterns let_{pm} is the same as let_p. Finally, we have added polymorphism annotations to let expressions (Fig. 4). Original **(Flat)** rule has been split into two, one for each kind of polymorphism. Although both behave the same from the point of view of values, the splitting is needed to guarantee type preservation. λ-abstractions have been omitted, since they are not supported by *let-rewriting*.

(Fapp) $f\ t_1\theta \ldots t_n\theta \ \rightarrow^l \ r\theta, \quad$ if $(f\ t_1 \ldots t_n \rightarrow r) \in \mathcal{P}$ and $\theta \in \mathcal{PS}ubst$

(LetIn) $e_1\ e_2 \rightarrow^l let_m\ X = e_2\ in\ e_1\ X, \quad$ if e_2 is an active expression, variable application, junk or *let* rooted expression, for X fresh.

(Bind) $let_K\ X = t\ in\ e\ \rightarrow^l\ e[X/t], \quad$ if $t \in Pat$

(Elim) $let_K\ X = e_1\ in\ e_2 \rightarrow^l e_2, \quad$ if $X \notin FV(e_2)$

(Flat$_m$) $let_m\ X = (let_K\ Y = e_1\ in\ e_2)\ in\ e_3\ \rightarrow^l\ let_K\ Y = e_1\ in\ (let_m\ X = e_2\ in\ e_3), \quad$ if $Y \notin FV(e_3)$

(Flat$_p$) $let_p\ X = (let_K\ Y = e_1\ in\ e_2)\ in\ e_3\ \rightarrow^l\ let_p\ Y = e_1\ in\ (let_p\ X = e_2\ in\ e_3)$ if $Y \notin FV(e_3)$

(LetAp) $(let_K\ X = e_1\ in\ e_2)\ e_3 \rightarrow^l let_K\ X = e_1\ in\ e_2\ e_3, \quad$ if $X \notin FV(e_3)$

(Contx) $\mathcal{C}[e] \rightarrow^l \mathcal{C}[e'], \ $ if $\mathcal{C} \neq [\ ]$, $e \rightarrow^l e'$ using any of the previous rules

where $K \in \{m, p\}$

Fig. 4. Higher order *let*-rewriting relation $\rightarrow^l$

Theorem 2 (Type preservation of the let transformation). *Assume $\mathcal{A} \vdash^\bullet e : \tau$ and let $\mathcal{P} \equiv \{\overline{f_{X_i}\ t_i \rightarrow X_i}\}$ be the rules of the projection functions needed in the transformation of e according to Fig. 3. Let also $\mathcal{A}'$ be the set of assumptions over that functions, defined as $\mathcal{A}' \equiv \{\overline{f_{X_i} : Gen(\tau_{X_i}, \mathcal{A})}\}$, where $\mathcal{A} \Vdash^\bullet \lambda t_i.X_i : \tau_{X_i}|\pi_{X_i}$. Then $\mathcal{A} \oplus \mathcal{A}' \vdash^\bullet TRL(e) : \tau$ and $wt_{\mathcal{A}\oplus\mathcal{A}'}(\mathcal{P})$.*

Th. 2 also states that the projection functions are well-typed. Then if we start from a well-typed program $\mathcal{P}$ wrt. $\mathcal{A}$ and apply the transformation to all its rules, the program extended with the projections rules will be well-typed wrt. the extended assumptions: $wt_{\mathcal{A}\oplus\mathcal{A}'}(\mathcal{P} \uplus \mathcal{P}')$. This result is straightforward, because $\mathcal{A}'$ does not contain any assumption for the symbols in $\mathcal{P}$, so $wt_{\mathcal{A}}(\mathcal{P})$ implies $wt_{\mathcal{A}\oplus\mathcal{A}'}(\mathcal{P})$.

Th. 3 states the subject reduction property for a *let-rewriting* step, but its extension to any number of steps is trivial.

Theorem 3 (Subject Reduction). *If $\mathcal{A} \vdash^\bullet e : \tau$ and $wt_{\mathcal{A}}(\mathcal{P})$ and $\mathcal{P} \vdash e \rightarrow^l e'$ then $\mathcal{A} \vdash^\bullet e' : \tau$.*

For this result to hold it is essential that the definition of well-typed program relies on $\vdash^\bullet$. A counterexample can be found in Ex. 1, where the program would be well-typed wrt. $\vdash$ but the subject reduction property fails for *and* (*cast* 0) *true* because of the rule for *co*.

The proof of the subject reduction property is based on the following Lemma, an important auxiliary result about the instantiation of transparent variables. Intuitively it states that if we have a pattern t with type τ and we change its variables by other expressions, the only way to obtain the same type τ for the substituted pattern is by changing the transparent variables for expressions with the same type. This is not guaranteed with opaque variables, and that is why we forbid their use in expressions.

Lemma 1. *Assume* $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau$*, where* $var(t) \subseteq \{\overline{X_i}\}$*. If* $\mathcal{A} \vdash t[\overline{X_i/s_i}] : \tau$ *and* X_j *is a transparent variable of* t *wrt.* $\mathcal{A}$ *then* $\mathcal{A} \vdash s_j : \tau_j$*.*

4 Type inference for expressions

The typing relation $\vdash^\bullet$ lacks some properties that prevent its usage as a type-checker mechanism in a compiler for a functional logic language. First, in spite of the syntax-directed style, the rules for $\vdash$ and $\vdash^\bullet$ have a bad operational behavior: at some steps they need to guess a type. Second, the types related to an expression can be infinite due to polymorphism. Finally, the typing relation needs all the assumptions for the symbols in order to work. To overcome this problems, type systems usually are accompanied with a type inference algorithm which returns a valid type for an expression and also establish the types for some symbols in the expression.

In this work we have given the type inference in Fig. 5 a relational style to show the similarities with the typing relation. But in essence, the inference rules represent an algorithm (similar to algorithm $\mathcal{W}$ [3,12]) which fails if any of the rules cannot be applied. This algorithm accepts a set of assumptions $\mathcal{A}$ and an expression e, and returns a simple type τ and a type substitution π. Intuitively, τ will be the "most general" type which can be given to e, and π the "minimum" substitution we have to apply to $\mathcal{A}$ in order to able to derive a type for e.

Th. 4 shows that the type and substitution found by the inference are correct, i.e., we can build a type derivation for the same type if we apply the substitution to the assumptions.

Theorem 4 (Soundness of $\Vdash^?$). $\mathcal{A} \Vdash^? e : \tau | \pi \Longrightarrow \mathcal{A}\pi \vdash^? e : \tau$

Th. 5 expresses the completeness of the inference process. If we can derive a type for an expression applying a substitution to the assumptions, then inference will succeed and will find a type and a substitution which are more general.

Theorem 5 (Completeness of $\Vdash$ wrt $\vdash$). *If* $\mathcal{A}\pi' \vdash e : \tau'$ *then* $\exists \tau, \pi, \pi''.\ \mathcal{A} \Vdash e : \tau | \pi \wedge \mathcal{A}\pi\pi'' = \mathcal{A}\pi' \wedge \tau\pi'' = \tau'$*.*

A result similar to Th. 5 cannot be obtained for $\Vdash^\bullet$ because of critical variables, as Example 3 shows.

Example 3 (Inexistence of a more general typing substitution). Let $\mathcal{A} \equiv \{snd' :: \alpha \to bool \to bool\}$ and consider the following two valid derivations $\mathcal{D}_1 \equiv \mathcal{A}[\alpha/bool] \vdash^\bullet \lambda(snd'\ X).X : (bool \to bool) \to bool$ and $\mathcal{D}_2 \equiv \mathcal{A}[\alpha/int] \vdash^\bullet \lambda(snd'\ X).X : (bool \to bool) \to int$. It is clear that there is not a substitution more general than $[\alpha/bool]$ and $[\alpha/int]$ which makes possible a type derivation for $\lambda(snd'\ X).X$. The only substitution more general than these two will be $[\alpha/\beta]$ (for some β), converting X in a critical variable.

In spite of this, we will see that $\Vdash^\bullet$ is still able to find a more general substitution when it exists. To formalize that, we will use the notion of $\Pi\!\!\Pi^\bullet_{\mathcal{A},e}$, which denotes the set collecting all type substitution π such that $\mathcal{A}\pi$ gives some type to e.

[iID]
$$\frac{}{\mathcal{A} \Vdash s : \tau | id} \quad if \begin{array}{l} s \in DC \cup FS \cup \mathcal{DV} \\ \wedge\ (s : \sigma) \in \mathcal{A} \wedge \sigma \succ_{var} \tau \end{array}$$

[iAPP]
$$\frac{\mathcal{A} \Vdash e_1 : \tau_1 | \pi_1 \quad \mathcal{A}\pi_1 \Vdash e_2 : \tau_2 | \pi_2}{\mathcal{A} \Vdash e_1\ e_2 : \alpha\pi | \pi_1\pi_2\pi} \quad if \begin{array}{l} \alpha\ fresh\ type\ variable \\ \wedge\ \pi = mgu(\tau_1\pi_2, \tau_2 \to \alpha) \end{array}$$

[iΛ]
$$\frac{\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_t | \pi_t \quad (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t \Vdash e : \tau | \pi}{\mathcal{A} \Vdash \lambda t.e : \tau_t\pi \to \tau | \pi_t\pi} \quad if \begin{array}{l} \{\overline{X_i}\} = var(t) \\ \wedge\ \overline{\alpha_i}\ fresh\ type\ variables \end{array}$$

[iLET$_m$]
$$\frac{\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_t | \pi_t \quad \mathcal{A}\pi_t \Vdash e_1 : \tau_1 | \pi_1 \quad (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_1\pi \Vdash e_2 : \tau_2 | \pi_2}{\mathcal{A} \Vdash \mathbf{let}_m\ t = e_1\ \mathbf{in}\ e_2 : \tau_2 | \pi_t\pi_1\pi\pi_2}$$
$$if\ \{\overline{X_i}\} = var(t) \wedge\ \overline{\alpha_i}\ fresh\ type\ variables \wedge\ \pi = mgu(\tau_t\pi_1, \tau_1)$$

[iLET$^X_{pm}$]
$$\frac{\mathcal{A} \Vdash e_1 : \tau_1 | \pi_1 \quad \mathcal{A}\pi_1 \oplus \{X : Gen(\tau_1, \mathcal{A}\pi_1)\} \Vdash e_2 : \tau_2 | \pi_2}{\mathcal{A} \Vdash \mathbf{let}_{pm}\ X = e_1\ \mathbf{in}\ e_2 : \tau_2 | \pi_1\pi_2}$$

[iLET$^h_{pm}$]
$$\frac{\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash h\ t_1 \ldots t_n : \tau_t | \pi_t \quad \mathcal{A}\pi_t \Vdash e_1 : \tau_1 | \pi_1 \quad (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_1\pi \Vdash e_2 : \tau_2 | \pi_2}{\mathcal{A} \Vdash \mathbf{let}_{pm}\ h\ t_1 \ldots t_n = e_1\ \mathbf{in}\ e_2 : \tau_2 | \pi_t\pi_1\pi\pi_2}$$
$$if\ h \in DC \cup FS \wedge\ \{\overline{X_i}\} = var(h\ t_1 \ldots t_n) \wedge\ \overline{\alpha_i}\ fresh\ type\ variables \wedge\ \pi = mgu(\tau_t\pi_1, \tau_1)$$

[iLET$_p$]
$$\frac{\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_t | \pi_t \quad \mathcal{A}\pi_t \Vdash e_1 : \tau_1 | \pi_1 \quad \mathcal{A}\pi_t\pi_1\pi \oplus \{\overline{X_i : Gen(\alpha_i\pi_t\pi_1\pi, \mathcal{A}\pi_t\pi_1\pi)}\} \Vdash e_2 : \tau_2 | \pi_2}{\mathcal{A} \Vdash \mathbf{let}_p\ t = e_1\ \mathbf{in}\ e_2 : \tau_2 | \pi_t\pi_1\pi\pi_2}$$
$$if\ \{\overline{X_i}\} = var(t) \wedge\ \overline{\alpha_i}\ fresh\ type\ variables \wedge\ \pi = mgu(\tau_t\pi_1, \tau_1)$$

[iP]
$$\frac{\mathcal{A} \Vdash e : \tau | \pi}{\mathcal{A} \Vdash^{\bullet} e : \tau | \pi} \quad if\ critVar_{\mathcal{A}\pi}(e) = \emptyset$$

Fig. 5. Inference rules

Definition 4 (Typing substitutions of e).
$\Pi^{\bullet}_{\mathcal{A},e} = \{\pi \in \mathcal{TS}ubst \mid \exists \tau \in SType.\ \mathcal{A}\pi \vdash^{\bullet} e : \tau\}$

Now we are ready to formulate our result regarding the maximality of $\Vdash^{\bullet}$.

Theorem 6 (Maximality of $\Vdash^{\bullet}$).

a) *$\Pi^{\bullet}_{\mathcal{A},e}$ has a maximum element $\Longleftrightarrow \exists \tau_g \in SType, \pi_g \in \mathcal{TS}ubst.\mathcal{A} \Vdash^{\bullet} e : \tau_g|\pi_g$.*

b) *If $\mathcal{A}\pi' \vdash^{\bullet} e : \tau'$ and $\mathcal{A} \Vdash^{\bullet} e : \tau|\pi$ then exists a type substitution π'' such that $\mathcal{A}\pi' = \mathcal{A}\pi\pi''$ and $\tau' = \tau\pi''$.*

5 Type inference for programs

In the functional programming setting, type inference does not need to distinguish between programs and expressions, because the program can be incorporated in the expression by means of let expressions and λ-abstractions. This way, the results given for expressions are also valid for programs. But in our framework it is different, because our semantics (*let-rewriting*) does not support λ-abstractions and our let expressions do not define new functions but only perform pattern matching. Thereby in our case we need to provide an explicit method for inferring the types of a whole program. By doing so, we will also provide a specification closer to implementations.

The type inference procedure for a program takes a set of assumptions $\mathcal{A}$ and a program $\mathcal{P}$ and returns a type substitution π. The set $\mathcal{A}$ must contain assumptions for all the symbols in the program, even for the functions defined in $\mathcal{P}$. We want to reflect the fact that in practice some defined functions may come with an explicit type declaration. Indeed this is a frequent way of documenting a program. Furthermore, type declarations are sometimes a real need, for instance if we want the language to support *polymorphic recursion* [15,9]. Therefore, for some of the functions –those for which we want to infer types– the assumption will be simply a fresh type variable, to be instantiated by the inference process. For the rest, the assumption will be a closed type-scheme, to be checked by the procedure.

Definition 5 (Type Inference of a Program). *The procedure $\mathcal{B}$ for type inference of a program $\{rule_1, \ldots, rule_m\}$ is defined as:*

$$\mathcal{B}(\mathcal{A}, \{rule_1, \ldots, rule_m\}) = \pi, \textit{if}$$

1. $\mathcal{A} \Vdash^{\bullet} (\varphi(rule_1), \ldots, \varphi(rule_m)) : (\tau_1, \ldots, \tau_m)|\pi$.
2. *Let $f^1 \ldots f^k$ be the function symbols of the rules $rule_i$ in $\mathcal{P}$ such that $\mathcal{A}(f^i)$ is a closed type-scheme, and τ^i the type obtained for $rule_i$ in step 1. Then τ^i must be a variant of $\mathcal{A}(f^i)$.*

φ is a transformation from rules to expressions defined as:

$$\varphi(f\ t_1 \ldots t_n \to e) = pair\ \lambda t_1 \ldots . \lambda t_n . e\ f$$

where $()$ *is the usual tuple constructor, with type* $() : \forall\overline{\alpha_i}.\alpha_1 \to \ldots \alpha_m \to (\alpha_1, \ldots, \alpha_m)$*; and* ***pair*** *is a special constructor of tuples of two elements of the same type, with type* $\mathbf{pair} :: \forall\alpha.\alpha \to \alpha \to \alpha$*.*

The procedure $\mathcal{B}$ has two important properties. It is sound: if the procedure $\mathcal{B}$ finds a substitution π then the program $\mathcal{P}$ is well typed with respect to the assumptions $\mathcal{A}\pi$ (Th. 7). And second, if the procedure $\mathcal{B}$ succeeds it finds a more general typing substitution (Th. 8). It is not true in general that the existence of a well-typing substitution π' implies the existence of a more general one. A counterexample of this fact is very similar to Ex. 3.

Theorem 7 (Soundness of $\mathcal{B}$). *If* $\mathcal{B}(\mathcal{A}, \mathcal{P}) = \pi$ *then* $wt_{\mathcal{A}\pi}(\mathcal{P})$*.*

Theorem 8 (Maximality of $\mathcal{B}$). *If* $wt_{\mathcal{A}\pi'}(\mathcal{P})$ *and* $\mathcal{B}(\mathcal{A}, \mathcal{P}) = \pi$ *then* $\exists\pi''$ *such that* $\mathcal{A}\pi' = \mathcal{A}\pi\pi''$*.*

Notice that types inferred for the functions are simple types. In order to obtain type-schemes we need and extra step of generalization, as discussed in the next section.

5.1 Stratified Inference of a Program

It is known that splitting a program into blocks of mutually recursive functions and inferring the types in order may reduce the need of providing explicit type-schemes. This situation is shown in Example 4.

Example 4 (Program Inference vs Stratified Inference).

$\mathcal{A} \equiv \{true : bool, 0 : int, id : \alpha, f : \beta, g : \gamma\}$
$\mathcal{P} \equiv \{id\ X \to X; f \to id\ true; g \to id\ 0\}$
$\mathcal{P}_1 \equiv \{id\ X \to X\}$, $\mathcal{P}_2 \equiv \{f \to id\ true\}$, $\mathcal{P}_3 \equiv \{g \to id\ 0\}$

An attempt to apply the procedure $\mathcal{B}$ to infer types for the whole program fails because it is not possible for id to have types $bool \to bool$ and $int \to int$ at the same time. We will need to provide explicitly the type-scheme for $id : \forall\alpha.\alpha \to \alpha$ in order to the type inference to succeed, yielding types $f : bool \to bool$ and $g : int \to int$. But this is not necessary if we first infer types for $\mathcal{P}_1$, obtaining $\delta \to \delta$ for id which will be generalized to $\forall\delta.\delta \to \delta$. With this assumption the type inference for both programs $\mathcal{P}_2$ and $\mathcal{P}_3$ will succeed with the expected types.

A general *stratified inference* procedure can be defined in terms of the basic inference $\mathcal{B}$. First, it calculates the graph of strongly connected components from the dependency graph of the program, using e.g. Kosaraju or Tarjan's algorithm [20]. Each strongly connected component will contain mutually dependent functions. Then it will infer types for every component (using $\mathcal{B}$) in topological order, generalizing the obtained types before following with the next component.

Although stratified inference needs less explicit type-schemes, programs involving polymorphic recursion still require explicit type-schemes in order to infer their types.

6 Conclusions and Future Work

In this paper we have proposed a type system for functional logic languages based on Damas & Milner type system. As far as we know, prior to our work only [5] treats with technical detail a type system for functional logic programming. Our paper makes clear contributions when compared to [5]:

- By introducing the notion critical variables, we are more liberal in the treatment of opaque variables, but still preserving the essential property of subject reduction; moreover, this liberality extends also to data constructors, dropping the traditional restriction of transparency required to them. This is somehow similar to what happens with *existential types* [13] or *generalized abstract datatypes* [8], a connection that we plan to further investigate in the future.
- Our type system considers local pattern bindings and λ-abstractions (also with patterns), that were missing in [5]. In addition to that, we have made a rather exhaustive analysis and formalization of different possibilities for polymorphism in local bindings.
- Subject reduction was proved in [5] wrt. a narrowing calculus. Here we do it wrt. an small-step operational semantics closer to real computations.
- In [5] programs came with explicit type declarations. Here we provide type inference algorithms where type declarations are optional.

We have in mind several lines for future work: apart from the relation to existential types mentioned above, we are interested in other known extensions of type system, like type classes or generic programming. We also want to generalize the subject reduction property to narrowing, using *let narrowing* reductions of [10], and taking into account known problems [5,1] in the interaction of HO narrowing and types. Handling extra variables (variables occurring only in right hand sides of rules) is another challenge from the viewpoint of types.

References

1. S. Antoy and A. P. Tolmach. Typed higher-order narrowing without higher-order strategies. In *Fuji International Symposium on Functional and Logic Programming*, pages 335–353, 1999.
2. B. Brassel. Post to the curry mailing list. `http://www.informatik.uni-kiel.de/~{}curry/listarchive/0706.html`, May 2008.
3. L. Damas and R. Milner. Principal type-schemes for functional programs. In *Proc. Symposium on Principles of Programming Languages*, pages 207–212, 1982.
4. J. González-Moreno, M. Hortalá-González, and M. Rodríguez-Artalejo. A higher order rewriting logic for functional logic programming. In *Proc. International Conference on Logic Programming (ICLP'97)*, pages 153–167. MIT Press, 1997.
5. J. González-Moreno, T. Hortalá-González, and Rodríguez-Artalejo, M. Polymorphic types in functional logic programming. In *Journal of Functional and Logic Programming*, volume 2001/S01, pages 1–71, 2001. Special issue of selected papers contributed to the International Symposium on Functional and Logic Programming (FLOPS'99).

6. M. Hanus. Multi-paradigm declarative languages. In *Proceedings of the International Conference on Logic Programming (ICLP 2007)*, pages 45–75. Springer LNCS 4670, 2007.
7. M. Hanus (ed.). Curry: An integrated functional logic language (version 0.8.2). Available at *http://www.informatik.uni-kiel.de/~curry/report.html*, March 2006.
8. S. L. P. Jones, D. Vytiniotis, S. Weirich, and G. Washburn. Simple unification-based type inference for gadts. In *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming, ICFP 2006*, pages 50–61. ACM, 2006.
9. A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. Type reconstruction in the presence of polymorphic recursion. *ACM Trans. Program. Lang. Syst.*, 15(2):290–311, 1993.
10. F. López-Fraguas, J. Rodríguez-Hortalá, and J. Sánchez-Hernández. Rewriting and call-time choice: the HO case. In *Proc. 9th International Symposium on Functional and Logic Programming (FLOPS'08)*, volume 4989 of *LNCS*, pages 147–162. Springer, 2008.
11. F. López-Fraguas and J. Sánchez-Hernández. $\mathcal{TOY}$: A multiparadigm declarative system. In *Proc. Rewriting Techniques and Applications (RTA'99)*, pages 244–247. Springer LNCS 1631, 1999.
12. L. M. Martins Damas. *Type Assignment in Programming Languages.* PhD thesis, University of Edinburgh, April 1985. Also appeared as Technical report CST-33-85.
13. J. C. Mitchell and G. D. Plotkin. Abstract types have existential type. *ACM Trans. Program. Lang. Syst.*, 10(3):470–502, 1988.
14. J. J. Moreno-Navarro, J. Mariño, A. del Pozo-Pietro, Á. Herranz-Nieva, and J. García-Martín. Adding type classes to functional-logic languages. In *1996 Joint Conf. on Declarative Programming, APPIA-GULP-PRODE'96*, pages 427–438, 1996.
15. A. Mycroft. Polymorphic type schemes and recursive definitions. In *Proceedings of the 6th Colloquium on International Symposium on Programming*, pages 217–228, London, UK, 1984. Springer-Verlag.
16. S. Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice Hall, 1987.
17. B. P. Pierce. *Advanced topics in types and programming languages.* MIT Press, Cambridge, MA, USA, 2005.
18. C. Reade. *Elements of Functional Programming.* Addison-Wesley, 1989.
19. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
20. R. Sedgewick. *Algorithms in C++, Part 5: Graph Algorithms*, chapter 19.8. Strong Components in Digraphs, pages 205–216. Addison-Wesley Professional, 2002.

A Proofs

Definition 6.
$\Pi_{\mathcal{A},e} = \{\pi \in \mathcal{TS}ubst \mid \exists \tau \in SType.\ \mathcal{A}\pi \vdash e : \tau\}$

Observation 1
Note that $\forall\overline{\alpha_i}.\tau = \forall\overline{\beta_i}.\tau[\overline{\alpha_i/\beta_i}]$ *if* $\{\overline{\beta_i}\} \cap FTV(\tau) = \emptyset$. *In other words, two different type-schemes are the same if we change the bounded variables for other variables which do not appear free in* τ. *For example,* $\forall\alpha,\beta.(\alpha,\beta) \to \alpha$ *is equal to* $\forall\gamma,\delta.(\gamma,\delta) \to \gamma$.

Observation 2
If $\sigma \succ \sigma'$ then $FTV(\sigma) \subseteq FTV(\sigma')$. It is clear from the definition of $\succ$. If α is a type variable in $FTV(\sigma)$ then it will not be affected by the substitution. Besides, α will be different from the generalized variables in σ'. Therefore $\alpha \in FTV(\sigma) \Longrightarrow \alpha \in FTV(\sigma')$, so $FTV(\sigma) \subseteq FTV(\sigma')$.

Observation 3
If $s \neq s'$ then $\mathcal{A} \oplus \{s : \sigma\} \oplus \{s' : \sigma'\}$ is the same as $\mathcal{A} \oplus \{s' : \sigma'\} \oplus \{s : \sigma\}$. This observation can be extended to sets of assumptions, in the sense that $\mathcal{A} \oplus \{\overline{X_i : \sigma_i}\} \oplus \{\overline{X'_j : \sigma'_j}\} = \mathcal{A} \oplus \{\overline{X'_j : \sigma'_j}\} \oplus \{\overline{X_i : \sigma_i}\}$ if $X_i \neq X'_j$ for all i and j.

Observation 4
If $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau$ then we can assume that $\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash e : \tau' | \pi$ such that $\mathcal{A}\pi = \mathcal{A}$.

Proof (Explanation). Intuitively, the inference finds a type which is more general than all the possible types for an expression, and also a type substitution which is necessary applying to the set of assumptions in order to derive a type for the expression. In this case it is possible from the original set of assumptions $\mathcal{A}$ to derive a type, so we do not need to change $\mathcal{A}$. Therefore the type substitution π from the inference would not need to affect $\mathcal{A}$, just only $\overline{\alpha_i}$ and the fresh variables generated during inference.

By Theorem 5 we know that there exists a type substitution π'' such that $\mathcal{A}\pi\pi'' = \mathcal{A}$ and $\tau'\pi'' = \tau$. This means that $\mathcal{A}\pi$ is just a renaming of some free type variables of $\mathcal{A}$, which are restored with the type substitution π''. Being $\mathcal{A}\pi$ a renaming of $\mathcal{A}$ is a consequence of the *mgu* algorithm used. In this case, during inference there will be some unifying steps between a free type variable α from $\mathcal{A}$ and a fresh one β. Clearly, both $[\alpha/\beta]$ and $[\beta/\alpha]$ are more general unifiers. In this cases if we choose the first, we will compute a substitution which will make $\mathcal{A}\pi$ a renaming of $\mathcal{A}$; but if we choose always to substitute the fresh type variables the set of assumption $\mathcal{A}\pi$ will remain the same as $\mathcal{A}$.

Observation 5
If $FTV(\mathcal{A}) = FTV(\mathcal{A}')$ then $Gen(\tau, \mathcal{A}) = Gen(\tau, \mathcal{A}')$

Observation 6 (Uniqueness of the type inference)
The result of a type inference is unique upon renaming of fresh type variables. In a type inference $\mathcal{A} \Vdash e : \tau | \pi$ the variables in $FTV(\tau)$, $Dom(\pi)$ or $Rng(\pi)$ which do not occur in $FTV(\mathcal{A})$ are fresh variables generated by the inference process, so the result will remain valid if we replace them with different fresh types variables.

Observation 7
In a type derivation $\mathcal{A} \vdash e : \tau$ will appear a type derivation for every subexpression e' of e. That is, the derivation will have a part of the tree rooted by $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e' : \tau'$, being τ' a suitable type for e', and being $\{\overline{X_i : \tau_i}\}$ a set

of assumptions over variables of the expression e which have been introduced by the rules $[\Lambda]$, $[LET_m], [LET^X_{pm}], [LET^h_{pm}]$ *or* $[LET_p]$.
If the expression is a pattern, the set of assumptions $\{\overline{X_i : \tau_i}\}$ *will be empty because the only rules used to type a pattern are* $[ID]$ *and* $[APP]$.

Observation 8
If $wt_{\mathcal{A}}(\mathcal{P})$ *and* $\mathcal{A}'$ *is a set of assumptions for variables, then* $wt_{\mathcal{A}\oplus\mathcal{A}'}(\mathcal{P})$.
The reason is that $\mathcal{A}'$ *does not change the assumptions for the function and constructor symbols in* $\mathcal{A}$. *Since there are not extra variables in the right hand sides, for every function rule in* $\mathcal{P}$ *the typing rule for the lambda expression will add assumptions for all the variables, shadowing the provided ones.*

Lemma 1
Assume $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau$, *where* $var(t) \subseteq \{\overline{X_i}\}$. *If* $\mathcal{A} \vdash t[\overline{X_i/s_i}] : \tau$ *and* X_j *is a transparent variable of* t *wrt.* $\mathcal{A}$ *then* $\mathcal{A} \vdash s_j : \tau_j$.

Proof. According to Observation 7, in the derivation of $\mathcal{A} \vdash t[\overline{X_i/s_i}] : \tau$ appear derivations for every subpattern s_i, and they have the form $\mathcal{A} \vdash s_i : \tau'_i$ for some τ'_i. We will prove that if X_j is a particular transparent variable of t, then $\tau_j = \tau'_j$. It is easy to see that taking the types τ'_i as assumptions for the original variables X_i we can construct a derivation of $\mathcal{A} \oplus \{X_i : \tau'_i\} \vdash t : \tau$, simply replacing the derivations for the subpatterns $\mathcal{A} \vdash s_i : \tau'_i$ with derivations for the variables $\mathcal{A} \oplus \{X_i : \tau'_i\} \vdash X_i : \tau'_i$ in the original derivation for $\mathcal{A} \vdash t[\overline{X_i/s_i}] : \tau$. Since X_j is a transparent variable of t wrt $\mathcal{A}$, by definition $\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_g|\pi_g$ and $FTV(\alpha_j\pi_g) \subseteq FTV(\tau_g)$. By Theorem 5, if any type for t can be derived from $\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\}\pi_s$ then π_g must be more general than π_s. We know that there are (at least) two substitutions π^1 and π^2 which can type t: $\pi^1 \equiv \{\overline{\alpha_i \mapsto \tau_i}\}$ and $\pi^2 \equiv \{\overline{\alpha_i \mapsto \tau'_i}\}$, so they must be more specific than π_g (i.e. there exist π, π' such that $\pi^1 = \pi_g\pi$ and $\pi^2 = \pi_g\pi'$. We also know (by Theorem 4) that $\mathcal{A} \oplus \{X_i : \alpha_i\} \Vdash t : \tau_g|\pi_g$ implies $(\mathcal{A} \oplus \{X_i : \alpha_i\})\pi_g \vdash t : \tau_g$, and by Theorem 1-a this implies that $(\mathcal{A} \oplus \{X_i : \alpha_i\})\pi_g\pi \vdash t : \tau_g\pi$; so $\tau_g\pi = \tau$ (the same thing happens with π': $\tau_g\pi' = \tau$).

At this point we can distinguish two cases:

A) X_j is transparent because of $FTV(\alpha_j\pi_g) = \emptyset$. Then $\tau_j = (\alpha_j\pi_g)\pi = \alpha_j\pi_g = (\alpha_j\pi_g)\pi' = \tau'_j$, because if $\alpha_j\pi_g$ does not have any free variable, it cannot be affected by any substitution.
B) X_j is transparent because of $FTV(\alpha_j\pi_g) \subseteq FTV(\tau_g)$. As $\tau_g\pi = \tau$ and $\tau_g\pi' = \tau$, then for every type variable β in $FTV(\tau_g)$ then $\beta\pi = \beta\pi'$. As every type variable β in $FTV(\alpha_j\pi_g)$ is also in $FTV(\tau_g)$ then as $\tau_j = (\alpha_j\pi_g)\pi = (\alpha_j\pi_g)\pi' = \tau'_j$.

□

Lemma 2.
If $\mathcal{A}\pi \Vdash e : \tau_1|\pi_1$ *then* $\exists\tau_2 \in SType, \pi_2\ \pi'' \in \mathcal{TS}ubst$ *s.t.* $\mathcal{A} \Vdash e : \tau_2|\pi_2$ *and* $\tau_2\pi'' = \tau_1$ *and* $\mathcal{A}\pi_2\pi'' = \mathcal{A}\pi\pi_1$.

Proof. By Theorem 4 $\mathcal{A}(\pi\pi_1) \Vdash e : \tau_1$. Then applying Theorem 5 $\mathcal{A} \Vdash e : \tau_2|\pi_2$ and there exists a type substitution $\pi'' \in \mathcal{TS}ubst$ such that $\tau_2\pi'' = \tau_1$ and $\mathcal{A}\pi_2\pi'' = \mathcal{A}\pi\pi_1$.

Lemma 3 (Equivalence of the two characterizations of opaque variable).
Let t be a pattern that admits type wrt. a given set of assumptions $\mathcal{A}$. Then

$$\exists \overline{\tau_i}, \tau \text{ s.t. } \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau \text{ and } FTV(\tau_i) \nsubseteq FTV(\tau)$$
$$\Longleftrightarrow$$
$$\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_g|\pi_g \text{ and } FTV(\alpha_i\pi_g) \nsubseteq FTV(\tau_g)$$

Proof.

– $\Longrightarrow$) The type derivation can be written as $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})[\overline{\alpha_i/\tau_i}] \vdash t : \tau$, so by Theorem 5 $\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_g|\pi_g$ and there exists some $\pi'' \in \mathcal{TS}ubst$ s.t. $\tau_g\pi'' = \tau$, $\mathcal{A}\pi_g\pi'' = \mathcal{A}$ and $\alpha_i\pi_g\pi'' = \tau_i$. We only need to prove that

$$FTV(\tau_i) \nsubseteq FTV(\tau) \Longrightarrow FTV(\alpha_i\pi_g) \nsubseteq FTV(\tau_g)$$

It is equivalent to prove

$$FTV(\alpha_i\pi_g) \subseteq FTV(\tau_g) \Longrightarrow FTV(\tau_i) \subseteq FTV(\tau)$$

which is trivial since $\alpha_i\pi_g\pi'' = \tau_i$ and $\tau_g\pi'' = \tau$, so

$$FTV(\alpha_i\pi_g) \subseteq FTV(\tau_g) \Longrightarrow FTV(\alpha_i\pi_g\pi'') \subseteq FTV(\tau_g\pi'')$$

– $\Longleftarrow$) By Theorem 4 $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_g \vdash t : \tau_g$, and $FTV(\alpha_i\pi_g) \nsubseteq FTV(\tau_g)$. Since t admits type by Observation 4 $\mathcal{A}\pi_g = \mathcal{A}$, so $\mathcal{A} \oplus \{\overline{X_i : \alpha_i\pi_g}\} \Vdash t : \tau_g$.

Lemma 4 (Decrease of opaque variables).
If $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau$ and $\mathcal{A}\pi \oplus \{\overline{X_i : \tau_i'}\} \vdash t : \tau'$ then $opaqueVar_{\mathcal{A}\pi}(t) \subseteq opaqueVar_{\mathcal{A}}(t)$.

Proof. Since $opaqueVar_{\mathcal{A}}(t) = var(t) \setminus transpVar_{\mathcal{A}}(e)$, then $opaqueVar_{\mathcal{A}\pi}(t) \subseteq opaqueVar_{\mathcal{A}}(t)$ is the same as $transpVar_{\mathcal{A}}(t) \subseteq transpVar_{\mathcal{A}\pi}(t)$. Then we have to prove that if a variable X_i of t is transparent wrt. $\mathcal{A}$ then it is also transparent wrt. $\mathcal{A}\pi$.

$\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}$ is the same as $\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\}[\overline{\alpha_i/\tau_i}]$, so by Theorem 5 we have that $\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_1|\pi_1$. Then the transparent variables of t will be those X_i such that $FTV(\alpha_i\pi_1) \subseteq FTV(\tau_1)$.

$\mathcal{A}\pi \oplus \{\overline{X_i : \tau_i'}\}$ is the same as $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi[\overline{\alpha_i/\tau_i'}]$, because we can assume that the variables $\overline{\alpha_i}$ does not appear in π. Then by Theorem 5 $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi \Vdash t : \tau_2|\pi_2$, and by Lemma 2 there exists a type substitution π'' such that $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi\pi_2 = (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_1\pi''$ and $\tau_2 = \tau_1\pi''$.

Therefore every data variable X_i which is transparent wrt. $\mathcal{A}$ will be also transparent wrt. $\mathcal{A}\pi$, because:

$$FTV(\alpha_i \pi_1) \subseteq FTV(\tau_1) \quad X_i \text{ is transparent wrt. } \mathcal{A}$$
$$FTV(\alpha_i \pi_1 \pi'') \subseteq FTV(\tau_1 \pi'') \text{ adding } \pi'' \text{ to both sides}$$
$$FTV(\alpha_i \pi \pi_2) \subseteq FTV(\tau_2) \quad X_i \text{ is transparent wrt. } \mathcal{A}\pi$$

Lemma 5. *If $\mathcal{A} \vdash \mathcal{C}[e] : \tau$ and in that derivation appear a derivation of the form $\mathcal{A} \oplus \mathcal{A}' \vdash e : \tau'$, and $\mathcal{A} \oplus \mathcal{A}' \vdash e' : \tau'$ then $\mathcal{A} \vdash \mathcal{C}[e'] : \tau$.*

Proof. We proceed by induction over the structure of the contexts:

[ ]) This case is straightforward because $[]e = e$ and $[]e' = e'$.

$\mathbf{e}_1$ $\mathcal{C}$) Since $(e_1\ \mathcal{C})[e] = e_1\ \mathcal{C}[e]$, if we have a derivation for $\mathcal{A} \vdash (e_1\ \mathcal{C})[e]$ it must be of the form:

$$[\mathbf{APP}]\ \frac{\mathcal{A} \vdash e_1 : \tau_1 \to \tau \qquad \mathcal{A} \vdash \mathcal{C}[e] : \tau_1}{\mathcal{A} \vdash e_1\ \mathcal{C}[e] : \tau}$$

A derivation of $\mathcal{A} \oplus \mathcal{A}' \vdash e : \tau'$ must appear in the whole derivation, so it must appear in the derivation $\mathcal{A} \vdash \mathcal{C}[e] : \tau_1$ (according to Observation 7). Since $\mathcal{A} \oplus \mathcal{A}' \vdash e' : \tau'$ then by the Induction Hypothesis we can state that $\mathcal{A} \vdash \mathcal{C}[e'] : \tau_1$, and we can construct a derivation for $\mathcal{A} \vdash (e_1\ \mathcal{C})[e']$:

$$[\mathbf{APP}]\ \frac{\mathcal{A} \vdash e_1 : \tau_1 \to \tau \qquad \mathcal{A} \vdash \mathcal{C}[e'] : \tau_1}{\mathcal{A} \vdash e_1\ \mathcal{C}[e'] : \tau}$$

$\mathcal{C}$ $\mathbf{e}_1$) Similar to the previous case.

$\mathbf{let}_m$ $\mathbf{X} = \mathcal{C}$ $\mathbf{in}$ $\mathbf{e}_1$) $(let_m\ X = \mathcal{C}\ in\ e_1)[e]$ is equal to $let_m\ X = \mathcal{C}[e]\ in\ e_1$, so a derivation of $\mathcal{A} \vdash (let_m\ X = \mathcal{C}\ in\ e_1)[e] : \tau$ must have the form:

$$[\mathbf{LET}_m]\ \frac{\mathcal{A} \oplus \{X : \tau_t\} \vdash X : \tau_t \qquad \mathcal{A} \vdash \mathcal{C}[e] : \tau_t \qquad \mathcal{A} \oplus \{X : \tau_t\} \vdash e_1 : \tau}{\mathcal{A} \vdash let_m\ X = \mathcal{C}[e]\ in\ e_1 : \tau}$$

Clearly, a derivation for $\mathcal{A} \oplus \mathcal{A}' \vdash e : \tau'$ will appear in the derivation for $\mathcal{A} \vdash \mathcal{C}[e] : \tau_t$ (Observation 7). Since $\mathcal{A} \oplus \mathcal{A}' \vdash e' : \tau'$ then by the Induction Hypothesis we can state that $\mathcal{A} \vdash \mathcal{C}[e'] : \tau_t$. With this information we can construct a derivation for (let$_m$ X = $\mathcal{C}$ in e_1)[e']:

$$[\mathbf{LET}_m]\ \frac{\mathcal{A} \oplus \{X : \tau_t\} \vdash X : \tau_t \qquad \mathcal{A} \vdash \mathcal{C}[e'] : \tau_t \qquad \mathcal{A} \oplus \{X : \tau_t\} \vdash e_1 : \tau}{\mathcal{A} \vdash let_m\ X = \mathcal{C}[e']\ in\ e_1 : \tau}$$

$\mathbf{let}_m$ $\mathbf{X} = e_1$ $\mathbf{in}$ $\mathcal{C}$) A type derivation of (let$_m$ X = e_1 in $\mathcal{C}$)[e] will have the form:

$$[\mathbf{LET}_m]\ \frac{\mathcal{A} \oplus \{X : \tau_t\} \vdash X : \tau_t \qquad \mathcal{A} \vdash e_1 : \tau_t \qquad \mathcal{A} \oplus \{X : \tau_t\} \vdash \mathcal{C}[e] : \tau}{\mathcal{A} \vdash let_m\ X = e_1\ in\ \mathcal{C}[e] : \tau}$$

By Observation 7, the derivation $\mathcal{A} \oplus \{X : \tau_t\} \vdash \mathcal{C}[e] : \tau$ will contain a derivation $(\mathcal{A} \oplus \{X : \tau_t\}) \oplus \mathcal{A}'' \vdash e : \tau'$. It is a premise that $(\mathcal{A} \oplus \{X : \tau_t\}) \oplus \mathcal{A}'' \vdash e' : \tau'$ (in this case $\mathcal{A}' = \{X : \tau_t\} \oplus \mathcal{A}''$), so by the Induction Hypothesis $\mathcal{A} \oplus \{X : \tau_t\} \vdash \mathcal{C}[e'] : \tau$ and we can construct a derivation $\mathcal{A} \vdash let_m\ X = e_1\ in\ \mathcal{C}[e'] : \tau$

$$[\mathbf{LET}_m]\ \frac{\mathcal{A} \oplus \{X : \tau_t\} \vdash X : \tau_t \quad \mathcal{A} \vdash e_1 : \tau_t \quad \mathcal{A} \oplus \{X : \tau_t\} \vdash \mathcal{C}[e'] : \tau}{\mathcal{A} \vdash let_m\ X = e_1\ in\ \mathcal{C}[e'] : \tau}$$

rest) The proofs for the cases $let_{pm}\ X = \mathcal{C}\ in\ e_1$, $let_{pm}\ X = e_1\ in\ \mathcal{C}$, $let_p\ X = \mathcal{C}\ in\ e_1$ and $let_p\ X = e_1\ in\ \mathcal{C}$ are similar to the proofs for let_m.

□

Lemma 6.
If $critVar_{\mathcal{A}}(e) = \emptyset$ *and* $critVar_{\mathcal{A}}(e') = \emptyset$ *then* $critVar_{\mathcal{A}}(e[X/e']) = \emptyset$.

Proof. We will proceed by induction over the structure of e.

Base Case

- c) Straightforward because $c[X/e'] = c$, so $critVar_{\mathcal{A}}(c[X/e']) = \emptyset$.
- f) The same as c.
- X) In this case $X[X/e'] = e'$, and $critVar_{\mathcal{A}}(e') = \emptyset$ from the premises.
- Y) Y is a variable distinct from X. Then $Y[X/e'] = Y$, so $critVar_{\mathcal{A}}(Y) = \emptyset$.

Induction Step

- $e_1\ e_2$) By definition $critVar_{\mathcal{A}}(e_1\ e_2) = \emptyset$ implies that $critVar_{\mathcal{A}}(e_1) = \emptyset$ and $critVar_{\mathcal{A}}(e_2) = \emptyset$. Then by the Induction Hypothesis $critVar_{\mathcal{A}}(e_1[X/e']) = \emptyset$ and $critVar_{\mathcal{A}}(e_2[X/e']) = \emptyset$. By definition $(e_1\ e_2)[X/e'] = e_1[X/e']\ e_2[X/e']$, so:

$$\begin{aligned} critVar_{\mathcal{A}}((e_1\ e_2)[X/e']) &= critVar_{\mathcal{A}}(e_1[X/e']\ e_2[X/e']) \\ &= critVar_{\mathcal{A}}(e_1[X/e']) \cup critVar_{\mathcal{A}}(e_2[X/e']) \\ &= \emptyset \cup \emptyset \\ &= \emptyset \end{aligned}$$

- $\lambda t.e$) We assume that $X \notin var(t)$ and $var(t) \cap FV(e') = \emptyset$. We know that $opaqueVar_{\mathcal{A}}(t) \cap FV(e) = \emptyset$ and $critVar_{\mathcal{A}}(e) = \emptyset$ from $critVar_{\mathcal{A}}(\lambda t.e) = \emptyset$. Moreover $opaqueVar_{\mathcal{A}}(t) \subseteq var(t)$, so $opaqueVar_{\mathcal{A}}(t) \cap FV(e') = \emptyset$. Since the interection of set is distributive, we have that $opaqueVar_{\mathcal{A}}(t) \cap (FV(e) \cup FV(e')) = (opaqueVar_{\mathcal{A}}(t) \cap FV(e)) \cup (opaqueVar_{\mathcal{A}}(t) \cap FV(e')) = \emptyset$. Since $FV(e[X/e']) \subseteq FV(e) \cup FV(e')$, then $opaqueVar_{\mathcal{A}}(t) \cap FV(e[X/e']) = \emptyset$. On the other hand by the Induction Hypothesis $critVar_{\mathcal{A}}(e[X/e']) = \emptyset$. Therefore

$$\begin{aligned} critVar_{\mathcal{A}}((\lambda t.e)[X/e']) &= critVar_{\mathcal{A}}(\lambda t.(e[X/e'])) \\ &= (opaqueVar_{\mathcal{A}}(t) \cap FV(e[X/e'])) \cup critVar_{\mathcal{A}}(e[X/e']) \\ &= \emptyset \cup \emptyset \\ &= \emptyset \end{aligned}$$

- $let_m\ t = e_1\ in\ e_2$) We assume that $X \notin var(t)$, $var(t) \cap FV(e') = \emptyset$, and $var(t) \cap FV(e_1) = \emptyset$. Since $critVar_{\mathcal{A}}(let_m\ t = e_1\ in\ e_2) = \emptyset$ then $opaqueVar_{\mathcal{A}}(t) \cap FV(e_2) = \emptyset$, $critVar_{\mathcal{A}}(e_1) = \emptyset$ and $critVar_{\mathcal{A}}(e_2) = \emptyset$. From $var(t) \cap FV(e') = \emptyset$ and $opaqueVar_{\mathcal{A}}(t) \subseteq var(t)$ we know that

$opaqueVar_{\mathcal{A}}(t) \cap FV(e') = \emptyset$. As in the previous case, $opaqueVar_{\mathcal{A}}(t) \cap (FV(e_2) \cup FV(e')) = \emptyset$ and $FV(e_2[X/e']) \subseteq FV(e_2) \cup FV(e')$, therefore $opaqueVar_{\mathcal{A}}(t) \cap FV(e_2[X/e']) = \emptyset$.
On the other hand by the Induction Hypothesis $critVar_{\mathcal{A}}(e_1[X/e']) = \emptyset$ and $critVar_{\mathcal{A}}(e_2[X/e']) = \emptyset$. Therefore

$$\begin{aligned} critVar_{\mathcal{A}}((let_m\ t = e_1\ in\ e_2)[X/e']) &= critVar_{\mathcal{A}}(let_m\ t = e_1[X/e']\ in\ e_2[X/e']) \\ &= (opaqueVar_{\mathcal{A}}(t) \cap FV(e_2[X/e'])) \cup \\ &\quad critVar_{\mathcal{A}}(e_1[X/e'])\ \cup\ critVar_{\mathcal{A}}(e_2[X/e']) \\ &= \emptyset \cup \emptyset \cup \emptyset \\ &= \emptyset \end{aligned}$$

The proofs for the let_{pm} and let_p cases are equal to the let_m case.

Lemma 7.

Let $\mathcal{A}$ be a set of assumptions, τ a type and $\pi \in \mathcal{TS}ubst$ such that for every type variable α which appears in τ and does not appear in $FTV(\mathcal{A})$ then $\alpha \notin Dom(\pi)$ and $\alpha \notin Rng(\pi)$. Then $(Gen(\tau, \mathcal{A}))\pi = Gen(\tau\pi, \mathcal{A}\pi)$.

Proof. We will study what happens with a type variable α of τ in both cases (types that are not variables are not modified by the generalization step).

- $\alpha \in FTV(\tau)$ and $\alpha \in FTV(\mathcal{A})$. In this case it cannot be generalized in $Gen(\tau, \mathcal{A})$, so in $(Gen(\tau, \mathcal{A}))\pi$ it will be transformed into $\alpha\pi$. Because $\alpha \in FTV(\mathcal{A})$, then all the variables in $\alpha\pi$ are in $FTV(\mathcal{A}\pi)$ and they cannot be generalized. Therefore in $Gen(\tau\pi, \mathcal{A}\pi)$ α will also be transformed into $\alpha\pi$.
- $\alpha \in FTV(\tau)$ and $\alpha \notin FTV(\mathcal{A})$. In this case α will be generalized in $Gen(\tau, \mathcal{A})$, and as π does not affect a generalized variable, it will remain in $(Gen(\tau, \mathcal{A}))\pi$. Because α is not in $Dom(\pi)$, then $\alpha\pi = \alpha$. $\alpha \notin Rng(\pi)$ and $\alpha \notin FTV(\mathcal{A})$, so it cannot appear in $\mathcal{A}\pi$. Therefore α will also be generalized in $Gen(\tau\pi, \mathcal{A}\pi)$.

□

Lemma 8 (Generalization and substitutions).

$$Gen(\tau, \mathcal{A})\pi \succ Gen(\tau\pi, \mathcal{A}\pi)$$

Proof. It is clear that if a type variable α in τ is not generalized in $Gen(\tau, \mathcal{A})$ (because it occurs in $FTV(\mathcal{A})$), then in the first type-scheme it will appear as $\alpha\pi$. In the second type scheme it will also appear as $\alpha\pi$ because all the variables in $\alpha\pi$ will be in $\mathcal{A}\pi$ (as $\alpha \in FTV(\mathcal{A}.)$). Therefore in every generic instance of the two type-schemes this part will be the same. On the other hand, if a type variable α is generalized in $Gen(\tau, \mathcal{A})$ then it will also appear generalized in $Gen(\tau, \mathcal{A})\pi$ (π will not affect it). It does not matter what happens with this part $\alpha\pi$ in $Gen(\tau\pi, \mathcal{A}\pi)$ because in every generic instance of $Gen(\tau, \mathcal{A})\pi$ the generalized α will be able to adopt all the types of any generic instance of the part $\alpha\pi$ in $Gen(\tau\pi, \mathcal{A}\pi)$. □

Lemma 9.

If $\mathcal{A} \Vdash^{\bullet} e : \tau|\pi$ *then* $\Pi_{\mathcal{A},e} = \Pi^{\bullet}_{\mathcal{A},e}$.

Proof. From definition of $\Vdash^{\bullet}$ we know that $\mathcal{A} \Vdash^{\bullet} e : \tau|\pi$. We need to prove that $\Pi_{\mathcal{A},e} \subseteq \Pi^{\bullet}_{\mathcal{A},e}$ and $\Pi^{\bullet}_{\mathcal{A},e} \subseteq \Pi_{\mathcal{A},e}$.

- $\Pi_{\mathcal{A},e} \subseteq \Pi^{\bullet}_{\mathcal{A},e}$) We prove that $\pi' \in \Pi_{\mathcal{A},e} \Longrightarrow \pi' \in \Pi^{\bullet}_{\mathcal{A},e}$. If $\pi' \in \Pi_{\mathcal{A},e}$ then $\mathcal{A}\pi' \vdash e : \tau'$, and by Theorem 5 there exists π'' such that $\mathcal{A}\pi' = \mathcal{A}\pi\pi''$ and $\tau' = \tau\pi''$. By Theorem 4 $\mathcal{A}\pi \vdash^{\bullet} e : \tau$, and by Theorem 1-a $\mathcal{A}\pi\pi'' \vdash^{\bullet} e : \tau\pi''$, which is equal to $\mathcal{A}\pi' \vdash^{\bullet} e : \tau\pi''$; so $\pi' \in \Pi^{\bullet}_{\mathcal{A},e}$.
- $\Pi^{\bullet}_{\mathcal{A},e} \subseteq \Pi_{\mathcal{A},e}$) From definition of $\Pi^{\bullet}_{\mathcal{A},e}$

□

Lemma 10.
$\mathcal{A} \vdash^{\bullet} e_1 : \tau_i, \ldots, \mathcal{A} \vdash^{\bullet} e_n : \tau_n \Longleftrightarrow \mathcal{A} \vdash^{\bullet} (e_1, \ldots, e_n) : (\tau_1, \ldots, \tau_n)$

Proof. Straightforward.

Theorem 1 (Properties of the typing relations).

a) If $\mathcal{A} \vdash^{?} e : \tau$ *then* $\mathcal{A}\pi \vdash^{?} e : \tau\pi$
b) Let s *be a symbol which does not appear in* e. *Then* $\mathcal{A} \vdash^{?} e : \tau \Longleftrightarrow \mathcal{A} \oplus \{s : \sigma_s\} \vdash^{?} e : \tau$.
c) If $\mathcal{A} \oplus \{X : \tau_x\} \vdash^{?} e : \tau$ *and* $\mathcal{A} \oplus \{X : \tau_x\} \vdash^{?} e' : \tau_x$ *then* $\mathcal{A} \oplus \{X : \tau_x\} \vdash^{?} e[X/e'] : \tau$.
d) If $\mathcal{A} \oplus \{s : \sigma\} \vdash e : \tau$ *and* $\sigma' \succ \sigma$, *then* $\mathcal{A} \oplus \{s : \sigma'\} \vdash e : \tau$.

Proof.
a.1) If $\mathcal{A} \vdash e : \tau$ *then* $\mathcal{A}\pi \vdash e : \tau\pi$
We prove it by induction over the size of the type derivation of $\mathcal{A} \vdash e : \tau$.

Base Case

- **[ID]** If we have a derivation of $\mathcal{A} \vdash s : \tau$ using **[ID]** is because τ is a generic instance of the type-scheme $\mathcal{A}(g) = \forall\overline{\alpha_i}.\tau'$. We can change this type-scheme by other equivalent $\forall\overline{\beta_i}.\tau''$ (according to Observation 1) where each variable β_i does not appear in $Dom(\pi)$ nor in $Rng(\pi)$. Then the generic instance τ will be of the form $\tau''[\overline{\beta_i/\tau_i}]$. We need to prove that $(\tau''[\overline{\beta_i/\tau_i}])\pi$ is a generic instance of $(\forall\overline{\beta_i}.\tau'')\pi$. Since π does not involve any variable β_i then $(\tau''[\overline{\beta_i/\tau_i}])\pi = \tau''\pi[\overline{\beta_i/\tau_i\pi}]$. Applying a substitution to a type-scheme is (by definition) applying it only to its free variables, but as no variable β_i appears in π then $(\forall\overline{\beta_i}.\tau'')\pi = \forall\overline{\beta_i}.(\tau''\pi)$. Then it is clear that $\tau''\pi[\overline{\beta_i/\tau_i\pi}]$ is a generic instance of $(\forall\overline{\beta_i}.\tau'')\pi$.

Induction Step
We have six different cases to consider accordingly to the inference rule used in the last step of the derivation.

– [**APP**] In this case we have a derivation

$$[\mathbf{APP}]\ \frac{\mathcal{A} \vdash e_1 : \tau_1 \to \tau \qquad \mathcal{A} \vdash e_2 : \tau_1}{\mathcal{A} \vdash e_1\ e_2 : \tau}$$

By the Induction Hypothesis $\mathcal{A}\pi \vdash e_1 : (\tau_1 \to \tau)\pi$ and $\mathcal{A}\pi \vdash e_2 : \tau_1\pi$. $(\tau_1 \to \tau)\pi \equiv \tau_1\pi \to \tau\pi$ so we can construct a derivation

$$[\mathbf{APP}]\ \frac{\mathcal{A}\pi \vdash e_1 : \tau_1\pi \to \tau\pi \qquad \mathcal{A}\pi \vdash e_2 : \tau_1\pi}{\mathcal{A} \vdash e_1\ e_2 : \tau\pi}$$

– [Λ] The derivation has the form

$$[\Lambda]\ \frac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t \qquad \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau}{\mathcal{A} \vdash \lambda t.e : \tau_t \to \tau}$$

By the Induction Hypothesis $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\})\pi \vdash \lambda t : \tau_t\pi$ and $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\})\pi \vdash e : \tau\pi$. But $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\})\pi \equiv \mathcal{A}\pi \oplus (\{\overline{X_i : \tau_i}\})\pi \equiv \mathcal{A}\pi \oplus \{\overline{X_i : \tau_i\pi}\}$ so we can build the type derivation

$$[\Lambda]\ \frac{\mathcal{A}\pi \oplus \{\overline{X_i : \tau_i\pi}\} \vdash t : \tau_t\pi \qquad \mathcal{A}\pi \oplus \{\overline{X_i : \tau_i\pi}\} \vdash e : \tau\pi}{\mathcal{A}\pi \vdash \lambda t.e : \tau_t \to \tau\pi}$$

– [$\mathbf{LET}_m$] The type derivation is

$$[\mathbf{LET}_m]\ \frac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t \qquad \mathcal{A} \vdash e_1 : \tau_t \qquad \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e_2 : \tau}{\mathcal{A} \vdash let_m\ t = e_1\ in\ e_2 : \tau}$$

By the Induction Hypothesis $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\})\pi \vdash t : \tau_t\pi$, $\mathcal{A}\pi \vdash e_1 : \tau_t\pi$ and $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\})\pi \vdash e_2 : \tau$. As in the previous case $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\})\pi \equiv \mathcal{A}\pi \oplus \{\overline{X_i : \tau_i\pi}\}$, so

$$[\mathbf{LET}_m]\ \frac{\begin{array}{c}\mathcal{A}\pi \oplus \{\overline{X_i : \tau_i\pi}\} \vdash t : \tau_t\pi \\ \mathcal{A}\pi \vdash e_1 : \tau_t\pi \\ \mathcal{A}\pi \oplus \{\overline{X_i : \tau_i\pi}\} \vdash e_2 : \tau\pi\end{array}}{\mathcal{A}\pi \vdash let_m\ t = e_1\ in\ e_2 : \tau\pi}$$

– [$\mathbf{LET}^X_{pm}$] The derivation will be

$$[\mathbf{LET}^X_{pm}]\ \frac{\mathcal{A} \vdash e_1 : \tau_x \qquad \mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A})\} \vdash e_2 : \tau}{\mathcal{A} \vdash let^X_{pm}\ X = e_1\ in\ e_2 : \tau}$$

First, we create a substitution π' that maps the variables of τ_x which do not appear in $FTV(\mathcal{A})$ to fresh variables which are not in $FTV(\mathcal{A})$ and do not occur in $Dom(\pi)$ nor in $Rng(\pi)$. Then by the Induction Hypothesis $\mathcal{A}\pi' \vdash e_1 : \tau_x\pi'$. Since π' does not contain in its domain any variable in $FTV(\mathcal{A})$, then $\mathcal{A}\pi' = \mathcal{A}$ and $\mathcal{A} \vdash e_1 : \tau_x\pi'$. π' only substitutes variables which do not appear in $\mathcal{A}$ by variables which are not in $\mathcal{A}$ either, so $Gen(\tau_x, \mathcal{A}) = Gen(\tau_x\pi', \mathcal{A})$. Then $\mathcal{A} \oplus \{X : Gen(\tau_x\pi', \mathcal{A})\} \vdash e_2 : \tau$ is a valid derivation, and by the Induction Hypothesis $(\mathcal{A} \oplus \{X : Gen(\tau_x\pi', \mathcal{A})\})\pi \vdash e_2 : \tau\pi$, which is the same that $\mathcal{A}\pi \oplus \{X : Gen(\tau_x\pi', \mathcal{A})\pi\} \vdash e_2 : \tau\pi$. By construction of π'

we know that for every variable of $\tau_x\pi'$ which does not appear in $\mathcal{A}$ it will not be in $Dom(\pi)$ nor in $Rng(\pi)$. Then we can apply Lemma 7 and we have that $\mathcal{A}\pi \oplus \{X : Gen(\tau_x\pi'\pi, \mathcal{A}\pi)\} \vdash e_2 : \tau\pi$. By the Induction Hypothesis over $\mathcal{A} \vdash e_1 : \tau_x\pi'$ we obtain $\mathcal{A}\pi \vdash e_1 : \tau_x\pi'\pi$. With this information we can construct a derivation

$$[\mathbf{LET}^X_{pm}]\ \frac{\mathcal{A}\pi \vdash e_1 : \tau_x\pi'\pi \qquad \mathcal{A}\pi \oplus \{X : Gen(\tau_x\pi'\pi, \mathcal{A}\pi)\} \vdash e_2 : \tau\pi}{\mathcal{A}\pi \vdash let^X_{pm}\ X = e_1\ in\ e_2 : \tau\pi}$$

- $[\mathbf{LET}^h_{pm}]$ Similar to the $[\mathrm{LET}_m]$ case.
- $[\mathbf{LET}_p]$ Similar to the $[\mathrm{LET}^X_{pm}]$ case, but instead of having to handle one single τ_x we need to handle a set of $\overline{\tau_i}$. The main idea is the same, creating a substitution π' to rename the variables of the $\overline{\tau_i}$ which do not appear in $\mathcal{A}$ and avoids their presence in the substitution π. Then we can apply Lemma 7 to all the generalizations and proceed as in the $[\mathrm{LET}^X_{pm}]$ case.

□

---

a.2) If $\mathcal{A} \vdash^\bullet e : \tau$ then $\mathcal{A}\pi \vdash^\bullet e : \tau\pi$

By definition of $\vdash^\bullet$ we know that $\mathcal{A} \vdash e : \tau$ and $critVar_{\mathcal{A}}(e) = \emptyset$. Then by Theorem 1-a $\mathcal{A}\pi \vdash e : \tau\pi$. To prove that $critVar_{\mathcal{A}\pi}(e) = \emptyset$ we use the decrease of opaque variables, stated in Lemma 4. From $\mathcal{A} \vdash e : \tau$ and $\mathcal{A}\pi \vdash e : \tau\pi$ we know that for every pattern t in e we have a derivation $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t$ and $\mathcal{A}\pi \oplus \{\overline{X_i : \tau'_i}\} \vdash t : \tau'$, being $\overline{X_i}$ the data variables in t. Then we can prove that $critVar_{\mathcal{A}\pi}(e) = \emptyset$ by induction over the structure of e.

Base Case

- s) $critVar_{\mathcal{A}\pi}(s) = \emptyset$ by definition.

Induction Step

- e_1e_2) By the Induction Hypothesis we have that $critVar_{\mathcal{A}\pi}(e_1) = \emptyset$ and $critVar_{\mathcal{A}\pi}(e_2) = \emptyset$, so $critVar_{\mathcal{A}\pi}(e_1e_2) = critVar_{\mathcal{A}\pi}(e_1) \cup critVar_{\mathcal{A}\pi}(e_2) = \emptyset \cup \emptyset = \emptyset$.
- $\lambda t.e$) By the Induction Hypothesis $critVar_{\mathcal{A}\pi}(e) = \emptyset$. $critVar_{\mathcal{A}}(t) = \emptyset$, so $(opaqueVar_{\mathcal{A}}(t) \cap var(t)) = \emptyset$. By Lemma 4 we know that $opaqueVar_{\mathcal{A}\pi}(t) \subseteq opaqueVar_{\mathcal{A}}(t)$, so $(opaqueVar_{\mathcal{A}\pi}(t) \cap var(t)) = \emptyset$. Then $critVar_{\mathcal{A}\pi}(\lambda t.e) = (opaqueVar_{\mathcal{A}\pi}(t) \cap var(t)) \cup critVar_{\mathcal{A}\pi}(e) = \emptyset \cup \emptyset = \emptyset$.
- $let_*\ t = e_1\ in\ e_2$) Similar to the previous case.

□

---

b.1)Let be s a symbol which does not appear in e. Then $\mathcal{A} \vdash e : \tau \iff \mathcal{A} \oplus \{s : \sigma_s\} \vdash e : \tau$.

$\Longrightarrow$) We will proceed by induction over the size of the derivation tree.
Base Case

– **[ID]** In this case the derivation will be:

$$[\textbf{ID}]\frac{}{\mathcal{A} \vdash s : \tau}$$

where $\mathcal{A}(g) \succ \tau$. If we add an assumption over a symbol different from s then $(\mathcal{A} \oplus \{s : \sigma_s\})(g) \succ \tau$, so

$$[\textbf{ID}]\frac{}{\mathcal{A} \oplus \{s : \sigma_s\} \vdash s : \tau}$$

Induction Step

– **[APP]** The derivation will have the form:

$$[\textbf{APP}]\frac{\mathcal{A} \vdash e_1 : \tau' \to \tau \qquad \mathcal{A} \vdash e_2 : \tau'}{\mathcal{A} \vdash e_1 e_2 : \tau}$$

By the Induction Hypothesis then $\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 : \tau' \to \tau$ and $\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_2 : \tau'$, therefore:

$$[\textbf{APP}]\frac{\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 : \tau' \to \tau \qquad \mathcal{A} \oplus \{s : \sigma_s\} \vdash e_2 : \tau'}{\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 e_2 : \tau}$$

– $[\Lambda]$ We have a type derivation

$$[\Lambda]\frac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau' \qquad \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau}{\mathcal{A} \vdash \lambda t.e : \tau' \to \tau}$$

By the Induction Hypothesis then $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\} \vdash t : \tau'$ and $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\} \vdash e : \tau$. s does not appear in $\lambda t.e$, so it will different from all the variables X_i and by Observation 3 $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\}$ is the same as $(\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{\overline{X_i : \tau_i}\}$. Therefore we can build a type derivation:

$$[\Lambda]\frac{(\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau' \qquad (\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau}{\mathcal{A} \oplus \{s : \sigma_s\} \vdash \lambda t.e : \tau' \to \tau}$$

– **[LET$_m$]** The type derivation will be:

$$[\textbf{LET}_m]\frac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t \qquad \mathcal{A} \vdash e_1 : \tau_t \qquad \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e_2 : \tau}{\mathcal{A} \vdash let_m\ t = e_1\ in\ e_2 : \tau}$$

By the Induction Hypothesis then $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\} \vdash t : \tau_t$, $\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 : \tau_t$ and $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\} \vdash e : \tau$. As in the previous case $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\} = (\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{\overline{X_i : \tau_i}\}$, so we can build a type derivation:

$$[\textbf{LET}_m]\frac{\begin{array}{c}(\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t \\ \mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 : \tau_t \\ (\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{\overline{X_i : \tau_i}\} \vdash e_2 : \tau\end{array}}{\mathcal{A} \oplus \{s : \sigma_s\} \vdash let_m\ t = e_1\ in\ e_2 : \tau}$$

– $[\mathbf{LET}^X_{pm}]$ The type derivation will be:

$$[\mathbf{LET}^X_{pm}]\frac{\mathcal{A} \vdash e_1 : \tau_x \quad \mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A})\} \vdash e_2 : \tau}{\mathcal{A} \vdash let_{pm}\ X = e_1\ in\ e_2 : \tau}$$

Here, $Gen(\tau_x, \mathcal{A})$ may be different from $Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma_s\})$. This is caused because there are some type variables $\overline{\alpha_i}$ in $FTV(\tau_x)$ such that they appear free in $\mathcal{A}$ but not in $\mathcal{A} \oplus \{s : \sigma_s\}$ (they appear only in a previous assumption for s in $\mathcal{A}$) or because there are some type variables $\overline{\beta_i}$ in $FTV(\tau_x)$ such that they do not occur free in $\mathcal{A}$ but they do appear free in $\mathcal{A} \oplus \{s : \sigma_s\}$ (they are added by σ_s). The first group of variables will be generalized in $Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma_s\})$ but not in $Gen(\tau_x, \mathcal{A})$. To handle the second group we can create a type substitution π from $\overline{\beta_i}$ to fresh type variables. This way $Gen(\tau_x\pi, \mathcal{A} \oplus \{s : \sigma_s\})$ will be a type-scheme more general than $Gen(\tau_x, \mathcal{A})$, and by Theorem 1-d then $\mathcal{A} \oplus \{X : Gen(\tau_x\pi, \mathcal{A} \oplus \{s : \sigma_s\})\} \vdash e_2 : \tau$. By Theorem 1-a we obtain the derivation $\mathcal{A}\pi \vdash e_1 : \tau_x\pi$, and since $\overline{\beta_i}$ are not in $Dom(\pi)$ then $\mathcal{A} \vdash e_1 : \tau_x\pi$. By the Induction Hypothesis $\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 : \tau_x\pi$ and $(\mathcal{A} \oplus \{X : Gen(\tau_x\pi, \mathcal{A} \oplus \{s : \sigma_s\})\}) \oplus \{s : \sigma_s\} \vdash e_2 : \tau$. As s is not in $let_m\ X = e_1\ in\ e_2$ then it is different from X, so $(\mathcal{A} \oplus \{X : Gen(\tau_x\pi, \mathcal{A} \oplus \{s : \sigma_s\})\}) \oplus \{s : \sigma_s\}$ is equal to $(\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{X : Gen(\tau_x\pi, \mathcal{A} \oplus \{s : \sigma_s\})\}$.
Therefore we can build the type derivation:

$$[\mathbf{LET}^X_{pm}]\frac{\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 : \tau_x\pi \quad (\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{X : Gen(\tau_x\pi, \mathcal{A} \oplus \{s : \sigma_s\})\} \vdash e_2 : \tau}{\mathcal{A} \oplus \{s : \sigma_s\} \vdash let_{pm}\ X = e_1\ in\ e_2 : \tau}$$

– $[\mathbf{LET}^h_{pm}]$ Similar to the $[\mathrm{LET}_m]$ case.
– $[\mathbf{LET}_p]$ Similar to the $[\mathrm{LET}^X_{pm}]$ case, creating a substitution π that solves the problem of the type variables which were generalized wrt. $\mathcal{A}$ but not wrt. $\mathcal{A} \oplus \{s : \sigma_s\}$.

$\Longleftarrow$) We will proceed again by induction over the size of the derivation tree.
Base Case
When the type derivation only applies the [ID] rule the proof is straightforward.

Induction Step

– [**APP**] The derivation will have the form:

$$[\mathbf{APP}]\frac{\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 : \tau' \to \tau \quad \mathcal{A} \oplus \{s : \sigma_s\} \vdash e_2 : \tau'}{\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 e_2 : \tau}$$

By the Induction Hypothesis then $\mathcal{A} \vdash e_1 : \tau' \to \tau$ and $\mathcal{A} \vdash e_2 : \tau'$, therefore:

$$[\mathbf{APP}]\frac{\mathcal{A} \vdash e_1 : \tau' \to \tau \quad \mathcal{A} \vdash e_2 : \tau'}{\mathcal{A} \vdash e_1 e_2 : \tau}$$

– [Λ] We have the type derivation:

$$[\Lambda]\frac{(\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau' \qquad (\mathcal{A} \oplus \{s : \sigma_s\})) \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau}{\mathcal{A} \oplus \{s : \sigma_s\} \vdash \lambda t.e : \tau' \to \tau}$$

Since s is not in $\lambda t.e$, s will be different from all the variables $\overline{X_i}$ and $(\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{\overline{X_i : \tau_i}\}$ will be the same as $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\}$. Having $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\} \vdash t : \tau'$ and $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma_s\} \vdash e : \tau$ we can apply the Induction Hypothesis and obtain $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau'$ and $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau$. With these two derivation we can build:

$$[\Lambda]\frac{\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau' \qquad \mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau}{\mathcal{A} \vdash \lambda t.e : \tau' \to \tau}$$

– [$\mathbf{LET}_m$] Similar to the [Λ] case.
– [$\mathbf{LET}^X_{pm}$] This case has to deal with the same problems as in [LET^X_{pm}] of the $\Longrightarrow$) case. We have a type derivation:

$$[\mathbf{LET}^X_{pm}]\frac{\begin{array}{c}\mathcal{A} \oplus \{s : \sigma_s\} \vdash e_1 : \tau_x \\ (\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{X : Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma_s\})\} \vdash e_2 : \tau\end{array}}{\mathcal{A} \oplus \{s : \sigma_s\} \vdash let_{pm}\ X = e_1\ in\ e_2 : \tau}$$

Again, the problem is that $Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma_s\})$ may not be the same as $Gen(\tau_x, \mathcal{A})$. As before, there may be variables $\overline{\alpha_i}$ in $FTV(\tau_x)$ which appear free in $\mathcal{A} \oplus \{s : \sigma_s\}$ but not in $\mathcal{A}$, and variables $\overline{\beta_i}$ in $FTV(\tau_x)$ which do not occur free in $\mathcal{A} \oplus \{s : \sigma_s\}$ but they do appear free in $\mathcal{A}$. The first group is not problematic, because they are variables which will be generalized in $Gen(\tau_x, \mathcal{A})$ but not in $Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma_s\})$. To solve the problem with the second group we create a type substitution π from $\overline{\beta}$ to fresh variables. This way $Gen(\tau_x\pi, \mathcal{A})$ will be a more general type-scheme than $Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma_s\})$. Applying Theorem 1-d then $(\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{X : Gen(\tau_x\pi, \mathcal{A})\} \vdash e_2 : \tau$. As s is different from X, then $(\mathcal{A} \oplus \{s : \sigma_s\}) \oplus \{X : Gen(\tau_x\pi, \mathcal{A})\}$ is the same as $(\mathcal{A} \oplus \{X : Gen(\tau_x\pi, \mathcal{A})\}) \oplus \{s : \sigma_s\}$, so the derivation $(\mathcal{A} \oplus \{X : Gen(\tau_x\pi, \mathcal{A})\}) \oplus \{s : \sigma_s\} \vdash e_2 : \tau$ is correct. Applying the Induction Hypothesis to this derivation we obtain $\mathcal{A} \oplus \{X : Gen(\tau_x\pi, \mathcal{A})\} \vdash e_2 : \tau$. By Theorem 1-a $(\mathcal{A} \oplus \{s : \sigma_s\})\pi \vdash e_1 : \tau_x\pi$, which is equal to $\mathcal{A} \oplus \{s : \sigma_s\pi\} \vdash e_1 : \tau_x\pi$ because $\overline{\beta_i}$ do not occur free in $\mathcal{A}$. Applying the Induction Hypothesis to this derivation, we obtain $\mathcal{A} \vdash e_1 : \tau_x\pi$. Therefore we can build the type derivation:

$$[\mathbf{LET}^X_{pm}]\frac{\mathcal{A} \vdash e_1 : \tau_x\pi \qquad \mathcal{A} \oplus \{X : Gen(\tau_x\pi, \mathcal{A})\} \vdash e_2 : \tau}{\mathcal{A} \vdash let_{pm}\ X = e_1\ in\ e_2 : \tau}$$

– [$\mathbf{LET}^h_{pm}$] Similar to the [Λ] case.
– [$\mathbf{LET}_p$] Similar to the [LET^X_{pm}] case.

□

---

b.2) Let be s a symbol which does not appear in e, and σ_s any type. Then $\mathcal{A} \vdash^\bullet e : \tau \iff \mathcal{A} \oplus \{s : \sigma_s\} \vdash^\bullet e : \tau$.

- $\Longrightarrow$) By definition of $\mathcal{A} \vdash^\bullet e : \tau$, $\mathcal{A} \vdash e : \tau$ and $critVar_{\mathcal{A}}(e) = \emptyset$. Since s does not occur in e by Theorem 1-b $\mathcal{A} \oplus \{s : \sigma_s\} \vdash e : \tau$. It will also be true that $critVar_{\mathcal{A} \oplus \{s:\sigma_s\}}(e) = \emptyset$ because the opaque variables in the patterns will not change by adding the new assumption, and neither the variables appearing in the rest of the expression. Therefore $\mathcal{A} \oplus \{s : \sigma_s\} \vdash^\bullet e : \tau$.
- $\Longleftarrow$) By definition of $\mathcal{A} \oplus \{s : \sigma_s\} \vdash^\bullet e : \tau$, $\mathcal{A} \oplus \{s : \sigma_s\} \vdash e : \tau$ and $critVar_{\mathcal{A} \oplus \{s:\sigma_s\}}(e) = \emptyset$. s does not appear in e, so by Theorem 1-b $\mathcal{A} \vdash e : \tau$. As in the previous case the critical variables of e will not change by deleting an assumption which is not used, so $\mathcal{A} \vdash^\bullet e : \tau$.

□

---

c.1) If $\mathcal{A} \oplus \{X : \tau_x\} \vdash e : \tau$ and $\mathcal{A} \oplus \{X : \tau_x\} \vdash e' : \tau_x$ then $\mathcal{A} \oplus \{X : \tau_x\} \vdash e[X/e'] : \tau$.

We will proceed by induction over the size of the expression e.

Base Case

- **[ID]** If $s \neq X$ then $s[X/e'] \equiv s$. On the contrary, if $s = X$ then the derivation will be:

$$[\mathbf{ID}]\frac{}{\mathcal{A} \oplus \{X : \tau_x\} \vdash X : \tau_x}$$

$X[X/e'] \equiv e'$, and the type derivation $\mathcal{A} \oplus \{X : \tau_x\} \vdash e' : \tau_x$ comes from the hypothesis.

Induction Step

- **[APP]** Just the application of the Induction Hypothesis.
- [Λ] We can assume that $\lambda t.e$ is such that the variables $\overline{X_i}$ in its pattern do not appear in $\mathcal{A} \oplus \{X : \tau_x\}$ nor in $FV(e')$. The derivation will have the form:

$$[\Lambda]\frac{\begin{array}{c}(\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau' \\ (\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau\end{array}}{\mathcal{A} \oplus \{X : \tau_x\} \vdash \lambda t.e : \tau' \to \tau}$$

As X is different from $\overline{X_i}$ then $(\lambda t.e)[X/e'] \equiv \lambda t.(e[X/e'])$, so the first derivation remains the same. We have from the hypothesis that $\mathcal{A} \oplus \{X : \tau_x\} \vdash e' : \tau_x$. Since none of the $\overline{X_i}$ appear in e' then by Theorem 1-b we can

add assumptions over that variables and obtain a derivation $(\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{\overline{X_i : \tau_i}\} \vdash e' : \tau_x$. Because $X \neq X_i$ for all i then by Observation 3 $(\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{\overline{X_i : \tau_i}\}$ is the same as $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{X : \tau_x\}$. We have $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{X : \tau_x\} \vdash e : \tau$ and $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{X : \tau_x\} \vdash e' : \tau_x$, so applying the Induction Hypothesis we obtain $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{X : \tau_x\} \vdash e[X/e'] : \tau$. Therefore we can build a new derivation:

$$[\Lambda]\frac{(\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau' \quad (\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{\overline{X_i : \tau_i}\} \vdash e[X/e'] : \tau}{\mathcal{A} \oplus \{X : \tau_x\} \vdash \lambda t.(e[X/e']) : \tau' \to \tau}$$

- $[\mathbf{LET}_m]$ The proof is similar to the $[\Lambda]$ case, provided that the variables of the pattern t do not occur in $FV(e')$ nor in $\mathcal{A} \oplus \{X : \tau_x\}$.
- $[\mathbf{LET}^X_{pm}]$ In this case Y is a fresh variable. The type derivation will be:

$$[\mathbf{LET}^X_{pm}]\frac{\mathcal{A} \oplus \{X : \tau_x\} \vdash e_1 : \tau_x \quad (\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{Y : Gen(\tau_x, \mathcal{A} \oplus \{X : \tau_x\})\} \vdash e_2 : \tau}{\mathcal{A} \oplus \{X : \tau_x\} \vdash let_{pm}\ Y = e_1\ in\ e_2 : \tau}$$

By the Induction Hypothesis $\mathcal{A} \oplus \{X : \tau_x\} \vdash e_1[X/e'] : \tau_x$. $X \neq Y$ and $Y \notin FV(e')$, so by Theorem 1-b we can add an assumption over the variable Y and get a derivation $(\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{Y : Gen(\tau_x, \mathcal{A} \oplus \{X : \tau_x\})\} \vdash e' : \tau_x$. By Observation 3 $(\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{Y : Gen(\tau_x, \mathcal{A} \oplus \{X : \tau_x\})\}$ is equal to $(\mathcal{A} \oplus \{Y : Gen(\tau_x, \mathcal{A} \oplus \{X : \tau_x\})\}) \oplus \{X : \tau_x\}$, so by the Induction Hypothesis $(\mathcal{A} \oplus \{Y : Gen(\tau_x, \mathcal{A} \oplus \{X : \tau_x\})\}) \oplus \{X : \tau_x\} \vdash e_2[X/e'] : \tau$. Again by Observation 3 $(\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{Y : Gen(\tau_x, \mathcal{A} \oplus \{X : \tau_x\})\} \vdash e_2[X/e'] : \tau$. Therefore we can construct a derivation:

$$[\mathbf{LET}^X_{pm}]\frac{\mathcal{A} \oplus \{X : \tau_x\} \vdash e_1[X/e'] : \tau_x \quad (\mathcal{A} \oplus \{X : \tau_x\}) \oplus \{Y : Gen(\tau_x, \mathcal{A} \oplus \{X : \tau_x\})\} \vdash e_2[X/e'] : \tau}{\mathcal{A} \oplus \{X : \tau_x\} \vdash let_{pm}\ Y = e_1[X/e']\ in\ e_2[X/e'] : \tau}$$

- $[\mathbf{LET}^h_{pm}]$ Equal to the $[\mathrm{LET}_m]$ case.
- $[\mathbf{LET}_p]$ The proof follows the same ideas as $[\mathrm{LET}_m]$ and $[\mathrm{LET}^X_{pm}]$.

□

---

c.2) If $\mathcal{A} \oplus \{X : \tau_x\} \vdash^\bullet e : \tau$ *and* $\mathcal{A} \oplus \{X : \tau_x\} \vdash^\bullet e' : \tau_x$ *then* $\mathcal{A} \oplus \{X : \tau_x\} \vdash^\bullet e[X/e'] : \tau$.

From the definition of $\vdash^\bullet$ we know that $\mathcal{A} \oplus \{X : \tau_x\} \vdash e : \tau$, $\mathcal{A} \oplus \{X : \tau_x\} \vdash e' : \tau_x$, $critVar_{\mathcal{A} \oplus \{X:\tau_x\}}(e) = \emptyset$ and $critVar_{\mathcal{A} \oplus \{X:\tau_x\}}(e') = \emptyset$. Then by Theorem 1-c $\mathcal{A} \oplus \{X : \tau_x\} \vdash e[X/e'] : \tau$. By Lemma 6 we also know that $critVar_{\mathcal{A} \oplus \{X:\tau_x\}}(e[X/e']) = \emptyset$, so by definition $\mathcal{A} \oplus \{X : \tau_x\} \vdash^\bullet e[X/e'] : \tau$. □

---

d.1) If $\mathcal{A} \oplus \{s : \sigma\} \vdash e : \tau$ *and* $\sigma' \succ \sigma$, *then* $\mathcal{A} \oplus \{s : \sigma'\} \vdash e : \tau$.

Base Case

– **[ID]** If $e \neq s$ then is trivial. If $e = s$ then the derivation will be:

$$[\textbf{ID}]\frac{}{\mathcal{A} \oplus \{s : \sigma\} \vdash s : \tau}$$

where $\sigma \succ \tau$. By Definition of generic instance, since $\sigma' \succ \sigma$ then $\sigma' \succ \tau$. So we can build the derivation:

$$[\textbf{ID}]\frac{}{\mathcal{A} \oplus \{s : \sigma'\} \vdash s : \tau}$$

Induction Step

– **[APP]** We have a type derivation:

$$[\textbf{APP}]\frac{\mathcal{A} \oplus \{s : \sigma\} \vdash e_1 : \tau' \to \tau \qquad \mathcal{A} \oplus \{s : \sigma\} \vdash e_2 : \tau'}{\mathcal{A} \oplus \{s : \sigma\} \vdash e_1 e_2 : \tau}$$

By the Induction Hypothesis we have that $\mathcal{A} \oplus \{s : \sigma'\} \vdash e_1 : \tau' \to \tau$ and $\mathcal{A} \oplus \{s : \sigma'\} \vdash e_2 : \tau'$. Then we can construct a type derivation with the more general assumptions:

$$[\textbf{APP}]\frac{\mathcal{A} \oplus \{s : \sigma'\} \vdash e_1 : \tau' \to \tau \qquad \mathcal{A} \oplus \{s : \sigma'\} \vdash e_2 : \tau'}{\mathcal{A} \oplus \{s : \sigma'\} \vdash e_1 e_2 : \tau}$$

– $[\Lambda]$ We can assume that s is different from all the variables $\overline{X_i}$. The type derivation will be:

$$[\Lambda]\frac{(\mathcal{A} \oplus \{s : \sigma\}) \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau' \qquad (\mathcal{A} \oplus \{s : \sigma\}) \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau}{\mathcal{A} \oplus \{s : \sigma\} \vdash \lambda t.e : \tau' \to \tau}$$

Since s is different from the variables $\overline{X_i}$, then $(\mathcal{A} \oplus \{s : \sigma\}) \oplus \{\overline{X_i : \tau_i}\}$ is the same as $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma\}$. Therefore $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma\} \vdash t : \tau'$ and $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma\} \vdash e : \tau$. By the Induction Hypothesis we have that $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma'\} \vdash t : \tau'$ and $(\mathcal{A} \oplus \{\overline{X_i : \tau_i}\}) \oplus \{s : \sigma'\} \vdash e : \tau$; and changing again the order in the assumptions we can build a derivation:

$$[\Lambda]\frac{(\mathcal{A} \oplus \{s : \sigma'\}) \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau' \qquad (\mathcal{A} \oplus \{s : \sigma'\}) \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau}{\mathcal{A} \oplus \{s : \sigma'\} \vdash \lambda t.e : \tau' \to \tau}$$

– $[\textbf{LET}_m]$ The proof is similar to the $[\Lambda]$ case.
– $[\textbf{LET}^X_{pm}]$ We assume that $s \neq X$. The type derivation is:

$$[\textbf{LET}^X_{pm}]\frac{\begin{array}{c}\mathcal{A} \oplus \{s : \sigma\} \vdash e_1 : \tau_x \\ (\mathcal{A} \oplus \{s : \sigma\}) \oplus \{X : Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma\})\} \vdash e_2 : \tau\end{array}}{\mathcal{A} \oplus \{s : \sigma\} \vdash let_{pm}\ X = e_1\ in\ e_2 : \tau}$$

By the Induction Hypothesis we have $\mathcal{A} \oplus \{s : \sigma'\} \vdash e_1 : \tau_x$. As $\sigma' \succ \sigma$ then by Observation 2 $FTV(\sigma') \subseteq FTV(\sigma)$. Therefore $FTV(\mathcal{A} \oplus \{s : \sigma'\}) = FTV(\mathcal{A}_s) \cup FTV(\sigma') \subseteq FTV(\mathcal{A}_s) \cup FTV(\sigma) = FTV(\mathcal{A} \oplus \{s : \sigma\})$, being $\mathcal{A}_s$ the result of deleting from $\mathcal{A}$ all the assumptions for the symbol s. With this information it is clear that $Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma'\}) \succ Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma\})$

because more variables could be generalized in $Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma'\})$. Then by the Induction Hypothesis $(\mathcal{A} \oplus \{s : \sigma\}) \oplus \{X : Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma'\})\} \vdash e_2 : \tau$. As $s \neq X$ then we can change the order of the assumptions and obtain a derivation $(\mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma'\})\}) \oplus \{s : \sigma\} \vdash e_2 : \tau$. Again by the Induction Hypothesis $(\mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma'\})\}) \oplus \{s : \sigma'\} \vdash e_2 : \tau$. With these derivations we can build the one we were trying to construct:

$$[\mathbf{LET}^X_{pm}] \frac{\mathcal{A} \oplus \{s : \sigma'\} \vdash e_1 : \tau_x \quad (\mathcal{A} \oplus \{s : \sigma'\}) \oplus \{X : Gen(\tau_x, \mathcal{A} \oplus \{s : \sigma'\})\} \vdash e_2 : \tau}{\mathcal{A} \oplus \{s : \sigma'\} \vdash let_{pm}\ X = e_1\ in\ e_2 : \tau}$$

– $[\mathbf{LET}^h_{pm}]$ Similar to the $[\Lambda]$ case.
– $[\mathbf{LET}_p]$ The proof is similar to the $[\mathrm{LET}^X_{pm}]$ case.

□

Theorem 2 (Type preservation of the let transformation).
Assume $\mathcal{A} \vdash^\bullet e : \tau$ and let $\mathcal{P} \equiv \{\overline{f_{X_i}\ t_i \to X_i}\}$ be the rules of the projection functions needed in the transformation of e according to Fig. 3. Let also $\mathcal{A}'$ be the set of assumptions over that functions, defined as $\mathcal{A}' \equiv \{\overline{f_{X_i} : Gen(\tau_{X_i}, \mathcal{A})}\}$, where $\mathcal{A} \Vdash^\bullet \lambda t_i.X_i : \tau_{X_i} | \pi_{X_i}$. Then $\mathcal{A} \oplus \mathcal{A}' \vdash^\bullet TRL(e) : \tau$ and $wt_{\mathcal{A} \oplus \mathcal{A}'}(\mathcal{P})$.

Proof. By structural induction over the expression e.

Base Case

– s) Straightforward.

Induction Step

– $e_1\ e_2$) We have the type derivation:

$$[\mathbf{APP}] \frac{\mathcal{A} \vdash e_1 : \tau_1 \to \tau \quad \mathcal{A} \vdash e_2 : \tau_1}{\mathcal{A} \vdash e_1\ e_2 : \tau}$$

Let be $\mathcal{A}^1$ and $\mathcal{A}^2$ the assumptions over the projection functions needed in e_1 and e_2 respectively. The by the Induction Hypothesis $\mathcal{A} \oplus \mathcal{A}^1 \vdash TRL(e_1)$ and $\mathcal{A} \oplus \mathcal{A}^2 \vdash TRL(e_2)$. Clearly the set of assumptions $\mathcal{A}'$ over the projection functions needed in the whole expression is $\mathcal{A}^1 \oplus \mathcal{A}^2$. Then by Theorem 1-b both derivations $\mathcal{A} \oplus \mathcal{A}' \vdash TRL(e_1)$ and $\mathcal{A} \oplus \mathcal{A}' \vdash TRL(e_2)$ are valid, and we can construct the type derivation:

$$[\mathbf{APP}] \frac{\mathcal{A} \oplus \mathcal{A}' \vdash TRL(e_1) : \tau_1 \to \tau \quad \mathcal{A} \oplus \mathcal{A}' \vdash TRL(e_2) : \tau_1}{\mathcal{A} \oplus \mathcal{A}' \vdash TRL(e_1)\ TRL(e_2) : \tau}$$

– $let_K\ X = e_1\ in\ e_2$) There are two cases, depending on the K:
$let_m\ X = e_1\ in\ e_2$:
The type derivation will be

$$[\mathbf{LET}_m]\frac{\begin{array}{c}\mathcal{A}\oplus\{X:\tau_t\}\vdash X:\tau_t\\ \mathcal{A}\vdash e_1:\tau_t\\ \mathcal{A}\oplus\{X:\tau_t\}\vdash e_2:\tau\end{array}}{\mathcal{A}\vdash let_m\ X=e_1\ in\ e_2:\tau}$$

By the Induction Hypothesis $\mathcal{A}\vdash TRL(e_1):\tau_t$ and $\mathcal{A}\oplus\{X:\tau_t\}\vdash TRL(e_2):\tau$. Then we can build the type derivation

$$[\mathbf{LET}_m]\frac{\begin{array}{c}\mathcal{A}\oplus\{X:\tau_t\}\vdash X:\tau_t\\ \mathcal{A}\vdash TRL(e_1):\tau_t\\ \mathcal{A}\oplus\{X:\tau_t\}\vdash TRL(e_2):\tau\end{array}}{\mathcal{A}\vdash let_m\ X=TRL(e_1)\ in\ TRL(e_2):\tau}$$

$\underline{let_p\ X=e_1\ in\ e_2}$:
The type derivation for the original expression is

$$[\mathbf{LET}_m]\frac{\begin{array}{c}\mathcal{A}\oplus\{X:\tau_t\}\vdash X:\tau_t\\ \mathcal{A}\vdash e_1:\tau_t\\ \mathcal{A}\oplus\{X:Gen(\tau_t,\mathcal{A})\}\vdash e_2:\tau\end{array}}{\mathcal{A}\vdash let_p\ X=e_1\ in\ e_2:\tau}$$

By the Induction Hypothesis $\mathcal{A}\vdash TRL(e_1):\tau_t$ and $\mathcal{A}\oplus\{X:Gen(\tau_t,\mathcal{A})\}\vdash TRL(e_2):\tau$. Then we can build the type derivation

$$[\mathbf{LET}_m]\frac{\begin{array}{c}\mathcal{A}\oplus\{X:\tau_t\}\vdash X:\tau_t\\ \mathcal{A}\vdash TRL(e_1):\tau_t\\ \mathcal{A}\oplus\{X:Gen(\tau_t,\mathcal{A})\}\vdash TRL(e_2):\tau\end{array}}{\mathcal{A}\vdash let_p\ X=TRL(e_1)\ in\ TRL(e_2):\tau}$$

– $let_{pm}\ X=e_1\ in\ e_2$) The type derivation for the original expression is

$$[\mathbf{LET}_{pm}]\frac{\begin{array}{c}\mathcal{A}\vdash e_1:\tau_t\\ \mathcal{A}\oplus\{X:Gen(\tau_t,\mathcal{A})\}\vdash e_2:\tau\end{array}}{\mathcal{A}\vdash let_{pm}\ X=e_1\ in\ e_2:\tau}$$

By the Induction Hypothesis $\mathcal{A}\vdash TRL(e_1):\tau_t$ and $\mathcal{A}\oplus\{X:Gen(\tau_t,\mathcal{A})\}\vdash TRL(e_2):\tau$. The type derivation $\mathcal{A}\oplus\{X:\tau_t\}\vdash X:\tau_t$ is trivial, so we can build the type derivation

$$[\mathbf{LET}_m]\frac{\begin{array}{c}\mathcal{A}\oplus\{X:\tau_t\}\vdash X:\tau_t\\ \mathcal{A}\vdash TRL(e_1):\tau_t\\ \mathcal{A}\oplus\{X:Gen(\tau_t,\mathcal{A})\}\vdash TRL(e_2):\tau\end{array}}{\mathcal{A}\vdash let_p\ X=TRL(e_1)\ in\ TRL(e_2):\tau}$$

– $let_m\ t=e_1\ in\ e_2$) In this case the original type derivation is:

$$[\mathbf{LET}_m]\frac{\begin{array}{c}\mathcal{A}\oplus\{\overline{X_i:\tau_i}\}\vdash t:\tau_t\\ \mathcal{A}\vdash e_1:\tau_t\\ \mathcal{A}\oplus\{\overline{X_i:\tau_i}\}\vdash e_2:\tau\end{array}}{\mathcal{A}\vdash let_m\ t=e_1\ in\ e_2:\tau}$$

It is easy to see that if $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau_t$ then $\mathcal{A} \vdash \lambda t.X_i : \tau_t \to \tau_i$. The assumptions over the projections functions in $\mathcal{A}'$ will be $\{\overline{f_{X_i} : Gen(\tau'_t \to \tau'_i, \mathcal{A})}\}$, where $\mathcal{A} \Vdash \lambda t.X_i : \tau'_t \to \tau'_i | \pi_{X_i}$. Since $\mathcal{A} \vdash \lambda t.X_i : \tau_t \to \tau_i$ we can assume that $\mathcal{A}\pi_{X_i} = \mathcal{A}$ (Observation 4), and by Theorem 5 we know that exists a type substitution π such that $\mathcal{A}\pi_{X_i}\pi = \mathcal{A}\pi = \mathcal{A}$ and $(\tau'_t \to \tau'_i)\pi = \tau_t \to \tau_i$. Therefore we can be sure that $Gen(\tau'_t \to \tau'_i, \mathcal{A}) \succ \tau_t \to \tau_i$, because π substitutes only the type variables in $\tau'_t \to \tau'_i$ which are generalized in $Gen(\tau'_t \to \tau'_i, \mathcal{A})$. If $\mathcal{A}'$ contains all the assumptions over the projection functions needed in the whole expression, it will contains assumptions over projection functions needed in e_1 ($\mathcal{A}^1$), e_2 ($\mathcal{A}^2$) and the pattern t ($\mathcal{A}^t \equiv \{\overline{f_{X_i} : Gen(\tau'_t \to \tau'_i, \mathcal{A})}\}$); so $\mathcal{A}' = \mathcal{A}^1 \oplus \mathcal{A}^2 \oplus \mathcal{A}^t$. Then we can build the type derivation:

$$[\mathbf{LET}_m] \frac{\begin{array}{c}\mathcal{A} \oplus \mathcal{A}' \oplus \{Y : \tau_t\} \vdash Y : \tau_t \\ \mathcal{A} \oplus \mathcal{A}' \vdash TRL(e_1) : \tau_t \\ \mathcal{A}_Y \vdash let_m\ X_1 = f_{X_1}\ Y\ in\ \ldots\ in\ TRL(e_2) : \tau\end{array}}{\mathcal{A} \oplus \mathcal{A}' \vdash let_m\ Y = TRL(e_1)\ in\ \overline{let_m\ X_i = f_{X_i}\ Y}\ in\ TRL(e_2) : \tau}$$

where the derivation $\mathcal{A}_Y \vdash let_m\ X_1 = f_{X_1}\ Y\ in\ \ldots\ in\ TRL(e_2) : \tau$ is

$$[\mathbf{LET}_m] \frac{[\mathbf{APP}]\ \dfrac{\mathcal{A}_Y \vdash f_{X_1} : \tau_t \to \tau_1 \quad \mathcal{A}_Y \vdash Y : \tau_t}{\mathcal{A}_Y \vdash f_{X_1}\ Y : \tau_1} \qquad [\mathbf{LET}_m]\ \dfrac{[\mathbf{ID}]\ \dfrac{}{\mathcal{A}_Y \oplus \{X_1 : \tau_1\} \vdash X_1 : \tau_1} \quad \mathcal{A}_Y \oplus \{\overline{X_i : \tau_i}\} \vdash TRL(e_2) : \tau}{\ldots}}{\mathcal{A}_Y \vdash let_m\ X_1 = f_{X_1}\ Y\ in\ \ldots\ in\ TRL(e_2) : \tau}$$

(being $\mathcal{A}_Y \equiv \mathcal{A} \oplus \mathcal{A}' \oplus \{Y : \tau_t\}$).
$\mathcal{A} \oplus \mathcal{A}' \oplus \{Y : \tau_t\} \vdash Y : \tau_t$ and $\mathcal{A}_Y \oplus \{X_1 : \tau_1\} \vdash X_1 : \tau_1$ are just the application of **[ID]** rule. By the Induction Hypothesis $\mathcal{A} \oplus \mathcal{A}^1 \vdash TRL(e_1) : \tau_t$, and by Theorem 1-b we can add the assumptions $\mathcal{A}^2 \oplus \mathcal{A}^t$, obtaining $\mathcal{A} \oplus \mathcal{A}' \vdash TRL(e_1) : \tau_t$. $\mathcal{A}_Y \vdash f_{X_1}\ Y : \tau_1$ is straightforward because $Gen(\tau'_t \to \tau'_i, \mathcal{A}\pi_{X_i}) \succ \tau_t \to \tau_i$ for all the projection functions. It is easy to see that this way the chain of let expressions will "collect" the same assumptions for the variables $\overline{X_i}$ that are introduced by the pattern in the original expression: $\{\overline{X_i : \tau_i}\}$. Then by the Induction Hypothesis $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \oplus \mathcal{A}^2 \vdash TRL(e_2) : \tau$, and by Theorem 1-b we can add the rest of the assumptions and obtain $\mathcal{A} \oplus \{\overline{X_i : \tau_i}\} \oplus \mathcal{A}^2 \oplus \mathcal{A}^1 \oplus \mathcal{A}^t \oplus \{Y : \tau_t\} \vdash TRL(e_2) : \tau$. Reorganizing the set of assumptions (since the symbols are all different), we obtain $\mathcal{A}_Y \oplus \{\overline{X_i : \tau_i}\} \vdash TRL(e_2) : \tau$.

- $let_{pm}\ t = e_1\ in\ e_2$) This case is equal to the previous one because the derivation of the original expression in both cases is the same (as t is a pattern we use $[\mathbf{LET}^h_{pm}]$, and this rule acts equal to $[\mathbf{LET}_m]$) and the transformed expressions are the same.
- $let_p\ t = e_1\ in\ e_2$) The type derivation will be:

$$[\mathbf{LET}_p]\frac{\begin{array}{c}\mathcal{A}\oplus\{\overline{X_i:\tau_i}\}\vdash t:\tau_t\\ \mathcal{A}\vdash e_1:\tau_t\\ \mathcal{A}\oplus\{\overline{X_i:Gen(\tau_i,\mathcal{A})}\}\vdash e_2:\tau\end{array}}{\mathcal{A}\vdash let_p\ t=e_1\ in\ e_2:\tau}$$

As in the previous case, $\mathcal{A}'$ will be $\{\overline{f_{X_i}:Gen(\tau'_t\to\tau'_i,\mathcal{A}\pi_{X_i})}\}$, where $\mathcal{A}\Vdash\lambda t.X_i:\tau'_t\to\tau'_i|\pi_{X_i}$. In addition, $\mathcal{A}\pi_{X_i}=\mathcal{A}$ (by the Observation 4), $Gen(\tau'_t\to\tau'_i,\mathcal{A})\succ\tau_t\to\tau_i$ and $\mathcal{A}'\equiv\mathcal{A}^1\oplus\mathcal{A}^2\oplus\mathcal{A}^t$. Then we can build a type derivation:

$$[\mathbf{LET}_p]\frac{\begin{array}{c}\mathcal{A}\oplus\mathcal{A}'\oplus\{Y:\tau_t\}\vdash Y:\tau_t\\ \mathcal{A}\oplus\mathcal{A}'\vdash TRL(e_1):\tau_t\\ \mathcal{A}'_1\vdash let_m\ X_1=f_{X_1}\ Y\ in\ \ldots\ in\ TRL(e_2):\tau\end{array}}{\mathcal{A}\oplus\mathcal{A}'\vdash let_p\ Y=TRL(e_1)\ in\ \overline{let_p\ X_i=f_{X_i}\ Y}\ in\ TRL(e_2):\tau}$$

where the derivation $\mathcal{A}_Y\vdash let_m\ X_1=f_{X_1}\ Y\ in\ \ldots\ in\ TRL(e_2):\tau$ is

$$[\mathbf{LET}_p]\frac{\begin{array}{cc}[\mathbf{APP}]\ \dfrac{\begin{array}{c}\mathcal{A}'_1\vdash f_{X_1}:\tau_t\to\tau_1\\ \mathcal{A}'_1\vdash Y:\tau_t\end{array}}{\mathcal{A}'_1\vdash f_{X_1}\ Y:\tau_1} & [\mathbf{LET}_p]\ \dfrac{\mathcal{A}'_{n+1}\vdash TRL(e_2):\tau}{\ldots}\end{array}}{\mathcal{A}'_1\vdash let_p\ X_1=f_{X_1}\ Y\ in\ \ldots\ in\ TRL(e_2):\tau}$$

being $\mathcal{A}'_1\equiv\mathcal{A}\oplus\mathcal{A}'\oplus\{Y:Gen(\tau_t,\mathcal{A}\oplus\mathcal{A}')\}$ and $\mathcal{A}'_i\equiv\mathcal{A}'_{i-1}\oplus\{X_{i-1}:Gen(\tau_{i-1},\mathcal{A}'_{i-1})\}$.
As in the previous case, all the derivations $\mathcal{A}'_i\vdash f_{X_i}\ Y:\tau_i$ are valid, because $\mathcal{A}'_i\vdash Y:\tau_t$. Notice that $Gen(\tau_t,\mathcal{A})=Gen(\tau_t,\mathcal{A}\oplus\mathcal{A}')$, as Observation 5 states, since $FTV(\mathcal{A})=FTV(\mathcal{A}\oplus\mathcal{A}')$. For the same reason, $Gen(\tau_i,\mathcal{A})=Gen(\tau_i,\mathcal{A}'_i)$, so the chain of let expressions will collect the same set of assumptions over the variables $\overline{X_i}$: $\{\overline{X_i:Gen(\tau_i,\mathcal{A})}\}$. By the Induction Hypothesis, we know that $\mathcal{A}\oplus\{\overline{X_i:Gen(\tau_i,\mathcal{A})}\}\oplus\mathcal{A}^2\vdash TRL(e_2):\tau$; and by Theorem 1-b we can add the assumptions $\mathcal{A}^1\oplus\mathcal{A}^t\oplus\{Y:Gen(\tau_t,\mathcal{A}\oplus\mathcal{A}')$ and obtain $\mathcal{A}\oplus\{\overline{X_i:Gen(\tau_i,\mathcal{A})}\}\oplus\mathcal{A}^2\oplus\mathcal{A}^1\oplus\mathcal{A}^t\oplus\{Y:Gen(\tau_t,\mathcal{A}\oplus\mathcal{A}')\}\vdash TRL(e_2):\tau$. Then reorganizing the assumptions we obtain $\mathcal{A}\oplus\mathcal{A}'\oplus\{Y:Gen(\tau_t,\mathcal{A}\oplus\mathcal{A}')\}\oplus\{\overline{X_i:Gen(\tau_i,\mathcal{A})}\}\vdash TRL(e_2):\tau$. Since $Gen(\tau_i,\mathcal{A})=Gen(\tau_i,\mathcal{A}'_i)$ then the previous derivation is equal to $\mathcal{A}'_{n+1}\vdash TRL(e_2):\tau$.

In all the cases it is true that $wt_{\mathcal{A}\oplus\mathcal{A}'}(\mathcal{P})$. Let X_i a data variable which is projected in the transformed expression, and t_i the compound pattern of a let expression where it appears. By Observation 7 we know that in the derivation $\mathcal{A}\vdash^\bullet e:\tau$ will appear a derivation $\mathcal{A}\oplus\mathcal{A}''\oplus\{X_i:\tau'_{X_i}\}\vdash t_i:\tau_i$ for a set of assumptions $\mathcal{A}''$ over some variables and X_i will not be opaque in t_i wrt. $\mathcal{A}\oplus\mathcal{A}''\oplus\{X_i:\tau'_{X_i}\}$. Then it is clear that $\mathcal{A}\vdash\lambda t_i.X_i:\tau_i\to\tau'_{X_i}$, and by Theorem 5 the type inference $\mathcal{A}\Vdash\lambda t_i.X_i:\tau_{X_i}|\pi_{X_i}$ will be correct. By Theorem 4 $\mathcal{A}\pi_{X_i}\vdash\lambda t_i.X_i:\tau_{X_i}$, and since by Observation 3 $\mathcal{A}\pi_{X_i}=\mathcal{A}$, then $\mathcal{A}\vdash\lambda t_i.X_i:$

τ_{X_i} is a valid derivation. Clearly X_i is not opaque in t_i wrt. $\mathcal{A}$, because only the assumptions for non variable symbols are used. Then $critVar_{\mathcal{A}}(\lambda t_i.X_i) = \emptyset$, so $\mathcal{A} \Vdash^{\bullet} \lambda t_i.X_i : \tau_{X_i} | \pi_{X_i}$ and $\mathcal{A} \vdash^{\bullet} \lambda t_i.X_i : \tau_{X_i}$. $\mathcal{A}'$ contains assumptions over projection functions, and they do not appear in $\lambda t_i.X_i$, so by Theorem 1-*b)* we can add these assumptions and obtain $\mathcal{A} \oplus \mathcal{A}' \vdash^{\bullet} \lambda t_i.X_i : \tau_{X_i}$. We know that in $\mathcal{A}'$ there will appear an assumption $\{f_{X_i} : Gen(\tau_{X_i}, \mathcal{A})\}$ for the projection function of the variable X_i, with rule $f_{X_i}\ t_i \to X_i$. We know that $FTV(\mathcal{A}) = FTV(\mathcal{A} \oplus \mathcal{A}')$ because since all the assumptions in $\mathcal{A}$ are of the form $Gen(\tau_{X_i}, \mathcal{A})$ they will not add any type variable, and since no f_{X_i} appears in $\mathcal{A}$ they will not shadow any assumption. Then τ_{X_i} will be a variant of $Gen(\tau_{X_i}, \mathcal{A})$.

Therefore for every data variable X_i which is projected then $\mathcal{A} \vdash \lambda t_i.X_i : \tau_{X_i}$ and τ_{X_i} is a variant of $\mathcal{A} \oplus \mathcal{A}'(f_{X_i}) = Gen(\tau_{X_i}, \mathcal{A})$, so all the program rules $f_{X_i}\ t_i \to X_i \in \mathcal{P}'$ are well-typed wrt. $\mathcal{A} \oplus \mathcal{A}'$ and $wt_{\mathcal{A} \oplus \mathcal{A}'}(\mathcal{P}')$.

Theorem 3 (Subject Reduction wrt $\vdash$).
If $\mathcal{A} \vdash e : \tau$ and $wt_{\mathcal{A}}(\mathcal{P})$ and $\mathcal{P} \vdash e \to^{l} e'$ then $\mathcal{A} \vdash e' : \tau$.

Proof. We proceed by case distinction over the rule of the let-rewriting relation $\to^{l}$ (Fig. 4) that we use to reduce e to e'.

– **(Fapp)** If we reduce an expression e using the **(Fapp)** rule is because e has the form $f\ t_1\theta \dots t_n\theta$ (being $f\ t_1 \dots t_n \to r$ a rule in $\mathcal{P}$ and $\theta \in \mathcal{PS}ubst$) and e' is $r\theta$. In this case we want to prove that $\mathcal{A} \vdash r\theta : \tau$. Since $wt_{\mathcal{A}}(\mathcal{P})$, then $\mathcal{A} \vdash^{\bullet} \lambda t_1 \dots \lambda t_n.r : \tau'_1 \to \dots \to \tau'_n \to \tau'$, being $\tau'_1 \to \dots \to \tau'_n \to \tau'$ a variant of $\mathcal{A}(f)$. We assume that the variables of the patterns $\overline{t_i}$ do not appear in $\mathcal{A}$ or in $Rng(\theta)$. The tree for this type derivation will be:

$$[\Lambda]\ \frac{\mathcal{A}_1 \vdash t_1 : \tau'_1 \qquad [\Lambda]\ \dfrac{\mathcal{A}_2 \vdash t_2 : \tau'_2 \qquad [\Lambda]\ \dfrac{\mathcal{A}_n \vdash t_n : \tau'_n \qquad \mathcal{A}_n \vdash r : \tau'}{\vdots}}{\mathcal{A}_2 \vdash t_3 \dots t_n : \tau'_3 \to \dots \to \tau'_n \to \tau'}}{\mathcal{A}_1 \vdash \lambda t_2 \dots t_n.r : \tau'_2 \to \dots \to \tau'_n \to \tau'}}{\mathcal{A} \vdash \lambda t_1 \dots t_n.r : \tau'_1 \to \tau'_2 \dots \to \tau'_n \to \tau'}$$

where $\mathcal{A}_j \equiv (\dots(\mathcal{A} \oplus \{\overline{X_{1i} : \tau''_{1i}}\}) \oplus \dots) \oplus \{\overline{X_{ji} : \tau''_{ji}}\}$ and X_{ji} is the i-th variable of the pattern t_j. We can write $\mathcal{A}_n$ as $\mathcal{A} \oplus \mathcal{A}'$, being $\mathcal{A}'$ the set of assumption over the variables of the patterns. As these variables are all different (the left hand side of the rules is linear), by Theorem 1-b we can add the rest of the assumptions to the $\mathcal{A}_j$ to get $\mathcal{A}_n$ and the derivation will remain valid, so $\forall j \in [1, n].\ \mathcal{A}_n \vdash t_j : \tau'_j$. Besides $critVar_{\mathcal{A}}(\lambda t_1 \dots \lambda t_n.r) = \emptyset$, so *a)* every variable X_{ji} which appears in r is transparent in the pattern t_j where it comes.
It is a premise that $\mathcal{A} \vdash f\ t_1\theta \dots t_n\theta : \tau$, and the tree of the type derivation will be:

$$\textbf{[APP]}\ \frac{\textbf{[APP]}\ \dfrac{\mathcal{A} \vdash f\ t_1\theta \dots t_{n-2}\theta : \tau_{n-1} \to (\tau_n \to \tau) \qquad \mathcal{A} \vdash t_{n-1}\theta : \tau_{n-1}}{\mathcal{A} \vdash f\ t_1\theta \dots t_{n-1}\theta : \tau_n \to \tau} \qquad \mathcal{A} \vdash t_n\theta : \tau_n}{\mathcal{A} \vdash f\ t_1\theta \dots t_n\theta : \tau}$$

where the type derivation $\mathcal{A} \vdash f\ t_1\theta \ldots t_{n-2}\theta : \tau_{n-1} \to (\tau_n \to \tau)$ is

$$[\textbf{APP}]\ \dfrac{\begin{array}{c}[\textbf{APP}]\ \dfrac{\begin{array}{c}[\textbf{ID}]\ \dfrac{}{\mathcal{A} \vdash f : \tau_1 \to \ldots \to \tau_n \to \tau \mathcal{A} \vdash f : \overline{\tau_n} \to \tau} \\ \mathcal{A} \vdash t_1\theta : \tau_1\end{array}}{\vdots}\end{array}}{\mathcal{A} \vdash f\ t_1\theta \ldots t_{n-2}\theta : \tau_{n-1} \to (\tau_n \to \tau)}$$

Because of that, we know that *b)* $\forall j \in [1, n]$. $\mathcal{A} \vdash t_j\theta : \tau_j$ and $\mathcal{A} \vdash f : \tau_1 \to \ldots \to \tau_n \to \tau$, being $\tau_1 \to \ldots \to \tau_n \to \tau$ a generic instance of the type $\mathcal{A}(f)$. Then there will exists a type substitution π such that $(\tau'_1 \to \ldots \to \tau'_n \to \tau')\pi = \tau_1 \to \ldots \to \tau_n \to \tau$, so $\forall j \in [1, n]$. $\tau'_j\pi = \tau_j$ and $\tau'\pi = \tau$. What is more, $Dom(\pi)$ does not contain any free type variable in $\mathcal{A}$, since π transforms a variant of the type of $\mathcal{A}(f)$ into a generic instance of the type of $\mathcal{A}(f)$. Then by Theorem 1-a $\mathcal{A}_n\pi \vdash t_j : \tau'_j\pi$, which is equal to *c)* $\mathcal{A} \oplus \mathcal{A}'\pi \vdash t_j : \tau'_j\pi$.
With *a)*, *b)* and *c)* and by Lemma 1 we can state that for every transparent variable X_{ji} in r then $\mathcal{A} \vdash X_{ji}\theta : \tau''_{ji}\pi$. None of the variables in $\mathcal{A}'$ appear in $X_{ji}\theta$, so by Theorem 1-b we can add these assumptions and obtain $\mathcal{A}_n \vdash X_{ji}\theta : \tau''_{ji}\pi$. According to the first derivation, we have $\mathcal{A}_n \vdash r : \tau'$. Here we can apply the Theorem 1-a again and get a derivation $\mathcal{A}_n\pi \vdash r : \tau'\pi$. Because $\mathcal{A}_n\pi \vdash X_{ji}\theta : \tau''_{ji}\pi$, then by Theorem 1-c $\mathcal{A}_n\pi \vdash r\theta : \tau'\pi$. As we have eliminated the variables in the expression, by Theorem 1-b we can delete their assumptions, obtaining a derivation $\mathcal{A}\pi \vdash r\theta : \tau'\pi$ (remember that $\mathcal{A}_n$ is $\mathcal{A} \oplus \mathcal{A}'$). And finally using the information we have about π, this derivation is equal to $\mathcal{A} \vdash r\theta : \tau$, the derivation we wanted to obtain.

– **(LetIn)** In this case $\mathcal{A} \vdash e_1e_2 : \tau$ and $\mathcal{P} \vdash e_1e_2 \to^l let_m\ X = e_2\ in\ e_1$. The type derivation of e_1e_2 will have the form:

$$[\textbf{APP}]\dfrac{\mathcal{A} \vdash e_1 : \tau_1 \to \tau \qquad \mathcal{A} \vdash e_2 : \tau_1}{\mathcal{A} \vdash e_1e_2 : \tau}$$

With this information we could build a type judgment for the let_m expression

$$[\textbf{LET}_m]\dfrac{\mathcal{A} \oplus \{X : \tau_1\} \vdash X : \tau_1 \qquad \mathcal{A} \vdash e_2 : \tau_1 \qquad [\textbf{APP}]\ \dfrac{\begin{array}{c}\mathcal{A} \oplus \{X : \tau_1\} \vdash e_1 : \tau_1 \to \tau \\ \mathcal{A} \oplus \{X : \tau_1\} \vdash X : \tau_1\end{array}}{\mathcal{A} \oplus \{X : \tau_1\} \vdash e_1X : \tau}}{\mathcal{A} \vdash let_m\ X = e_2\ in\ e_1X : \tau}$$

$\mathcal{A} \oplus \{X : \tau_1\} \vdash X : \tau_1$ is a valid derivation because is an application of the [ID] rule. And since X is a fresh variable, by Theorem 1-b we can add the assumption and obtain $\mathcal{A} \oplus \{X : \tau_1\} \vdash e_1 : \tau_1 \to \tau$.

– **(Bind)** We will distinguish between the let_m and the let_p case. In both cases we assume that the variable X is fresh.

let_m) In the let_m case the type derivation will have the form:

$$[\textbf{LET}_m]\dfrac{\mathcal{A} \oplus \{X : \tau_t\} \vdash X : \tau_t \qquad \mathcal{A} \vdash t : \tau_t \qquad \mathcal{A} \oplus \{X : \tau_t\} \vdash e : \tau}{\mathcal{A} \vdash let_m\ X = t\ in\ e : \tau}$$

As X is different from all the variables $\overline{X_i}$ of the pattern t, then by Theorem 1-b we can add the assumption over the variable X and obtain the derivation $\mathcal{A} \oplus \{X : \tau_t\} \vdash t : \tau_t$. Applying the Theorem 1-c then $\mathcal{A} \oplus \{X : \tau_t\} \vdash e[X/t] : \tau$. X will not appear in $e[X/t]$, so again by Theorem 1-b we can eliminate the assumption, concluding that $\mathcal{A} \vdash e[X/t] : \tau$.

let_p) Here the type derivations will be:

$$[\mathbf{LET}_p]\frac{\mathcal{A} \oplus \{X : \tau_t\} \vdash X : \tau_t \quad \mathcal{A} \vdash t : \tau_t \quad \mathcal{A} \oplus \{X : Gen(\tau_t, \mathcal{A})\} \vdash e : \tau}{\mathcal{A} \vdash let_p\ X = t\ in\ e : \tau}$$

and we want to prove that $\mathcal{A} \vdash e[X/t] : \tau$. We have a type derivation for $\mathcal{A} \oplus \{X : Gen(\tau_t, \mathcal{A})\} \vdash e : \tau$, and according to Observation 7 there will be derivations $(\mathcal{A} \oplus \{X : Gen(\tau_t, \mathcal{A})\}) \oplus \mathcal{A}'_i \vdash X : \tau_i$ for every appearance of X in e. In these cases, $\mathcal{A}'_i$ will only contain assumptions over variables $\overline{X_i}$ in let or lambda expressions of e. Suppose that all these variables have been renamed to fresh variables. We can create a type substitution π from the variables $\overline{\alpha_i}$ of τ_t which do not appear in $\mathcal{A}$ to fresh type variables $\overline{\beta_i}$. It is clear that $Gen(\tau_t, \mathcal{A})$ is equivalent to $Gen(\tau_t\pi, \mathcal{A})$, so $\mathcal{A} \oplus \{X : Gen(\tau_t\pi, \mathcal{A})\} \vdash e : \tau$ is a valid derivation. By Theorem 1-a $\mathcal{A}\pi \vdash t : \tau_t\pi$, and since $\overline{\alpha_i}$ are not in $\mathcal{A}$ then $\mathcal{A} \vdash t : \tau_t\pi$. X and $\overline{X_i}$ are fresh so they do not appear in t and by Theorem 1-b we can add assumptions to the derivation $\mathcal{A} \vdash t : \tau_t\pi$, obtaining $(\mathcal{A} \oplus \{X : Gen(\tau_t\pi, \mathcal{A})\}) \oplus \mathcal{A}'_i \vdash t : \tau_t\pi$. The types τ_i will be generic instances of $Gen(\tau_t, \mathcal{A})$, and also of $Gen(\tau_t\pi, \mathcal{A})$. Then for each τ_i there will exist a type substitution π'_i from the generalized variables $\overline{\beta_i}$ in $Gen(\tau_t\pi, \mathcal{A})$ to types that will hold $\tau_t\pi\pi'_i \equiv \tau_i$. By Theorem 1-a we can convert $(\mathcal{A} \oplus \{X : Gen(\tau_t\pi, \mathcal{A})\}) \oplus \mathcal{A}'_i \vdash t : \tau_t\pi$ into $((\mathcal{A} \oplus \{X : Gen(\tau_t\pi, \mathcal{A})\}) \oplus \mathcal{A}'_i)\pi'_i \vdash t : \tau_t\pi\pi'_i$, and as $\overline{\beta_i}$ are fresh variables then $(\mathcal{A} \oplus \{X : Gen(\tau_t\pi, \mathcal{A})\}) \oplus \mathcal{A}'_i \vdash t : \tau_t\pi\pi'_i$ (note that π'_i does not affect $Gen(\tau_t\pi, \mathcal{A})$ because the variables $\overline{\beta_i}$ are generalized). This way in every place of the original derivation where we have $(\mathcal{A} \oplus \{X : Gen(\tau_t, \mathcal{A})\}) \oplus \mathcal{A}'_i \vdash X : \tau_i$ we could place a derivation $(\mathcal{A} \oplus \{X : Gen(\tau_t\pi, \mathcal{A})\}) \oplus \mathcal{A}'_i \vdash t : \tau_i$. The resulting expression of this substitution will be $e[X/t]$, so $\mathcal{A} \oplus \{X : Gen(\tau_t\pi, \mathcal{A})\} \vdash e[X/t] : \tau$. It is clear that X does not appear in $e[X/t]$, so by Theorem 1-b we can eliminate the assumption over the X and obtain a derivation $\mathcal{A} \vdash e[X/t] : \tau$, as we wanted to prove.

- **(Elim)** In this case it does not matter what type of let expression it was (let_m or let_p). The rewriting step will be of the form $\mathcal{P} \vdash let_*\ X = e_1\ in\ e_2 \to^l e_2$. The type derivation of $\mathcal{A} \vdash let_*\ X = e_1\ in\ e_2 : \tau$ will have a branch $\mathcal{A} \oplus \{X : \sigma'\} \vdash e_2 : \tau$ for some σ. Since we are using the **(Elim)** rule, X does not appear in e_2 so by Theorem 1-b we can derive the same type eliminating that assumption, obtaining $\mathcal{A} \vdash e_2 : \tau$.
- **(Flat$_m$)** There are two cases, depending on the second let expression. In both cases we assume that $X \neq Y$.
  - $\mathcal{P} \vdash let_m\ X = (let_m\ Y = e_1\ in\ e_2)\ in\ e_3 \to^l let_m\ Y = e_1\ in\ (let_m\ X = e_2\ in\ e_3)$.

The type derivation will be:

$$[\mathbf{LET}_m]\dfrac{\dfrac{\mathcal{A}\oplus\{Y:\tau_y\}\vdash Y:\tau_y \quad \mathcal{A}\vdash e_1:\tau_y \quad \mathcal{A}\oplus\{Y:\tau_y\}\vdash e_2:\tau_x}{\mathcal{A}\vdash let_m\ Y=e_1\ in\ e_2:\tau_x}[\mathbf{LET}_m] \quad \mathcal{A}\oplus\{X:\tau_x\}\vdash X:\tau_x \quad \mathcal{A}\oplus\{X:\tau_x\}\vdash e_3:\tau}{\mathcal{A}\vdash let_m\ X=(let_m\ Y=e_1\ in\ e_2)\ in\ e_3:\tau}$$

Then we can build a type derivation

$$[\mathbf{LET}_m]\dfrac{\dfrac{(\mathcal{A}\oplus\{Y:\tau_y\})\oplus\{X:\tau_x\}\vdash X:\tau_x \quad \mathcal{A}\oplus\{Y:\tau_y\}\vdash e_2:\tau_x \quad (\mathcal{A}\oplus\{Y:\tau_y\})\oplus\{X:\tau_x\}\vdash e_3:\tau}{\mathcal{A}\oplus\{Y:\tau_y\}\vdash let_m\ X=e_2\ in\ e_3:\tau}[\mathbf{LET}_m] \quad \mathcal{A}\oplus\{Y:\tau_y\}\vdash Y:\tau_y \quad \mathcal{A}\vdash e_1:\tau_y}{\mathcal{A}\vdash let_m\ Y=e_1\ in\ (let_m\ X=e_2\ in\ e_3):\tau}$$

The only two derivations which do not come from the hypotheses are $(\mathcal{A}\oplus\{Y:\tau_y\})\oplus\{X:\tau_x\}\vdash X:\tau_x$ and $(\mathcal{A}\oplus\{Y:\tau_y\})\oplus\{X:\tau_x\}\vdash e_3:\tau$. The first is the application of the **[ID]** rule. From the hypotheses we have a derivation $\mathcal{A}\oplus\{X:\tau_x\}\vdash e_3:\tau$. Since we are rewriting using the **(Flat)** rule, we are sure that Y is not in e_3 and by Theorem 1-b we can add the assumption over the Y, obtaining the derivation $(\mathcal{A}\oplus\{X:\tau_x\})\oplus\{Y:\tau_y\}\vdash e_3:\tau$. X is different from Y, so according to Observation 3 $(\mathcal{A}\oplus\{X:\tau_x\})\oplus\{Y:\tau_y\}$ is the same as $(\mathcal{A}\oplus\{Y:\tau_y\})\oplus\{X:\tau_x\}$. Therefore $(\mathcal{A}\oplus\{Y:\tau_y\})\oplus\{X:\tau_x\}\vdash e_3:\tau$ is a valid derivation.

- $\mathcal{P}\vdash let_m\ X=(let_p\ Y=e_1\ in\ e_2)\ in\ e_3\rightarrow^l let_p\ Y=e_1\ in\ (let_m\ X=e_2\ in\ e_3)$. Similar to the previous case.

– **(Flat$_p$)** We will treat the two different cases:

- $\mathcal{P}\vdash let_p\ X=(let_p\ Y=e_1\ in\ e_2)\ in\ e_3\rightarrow^l let_p\ Y=e_1\ in\ (let_p\ X=e_2\ in\ e_3)$.

  The type derivation of the original expression is (being $\mathcal{A}_Y\equiv\mathcal{A}\oplus\{Y:Gen(\tau_y,\mathcal{A})\}$)

$$[\mathbf{LET}_p]\dfrac{\dfrac{\mathcal{A}\oplus\{Y:\tau_y\}\vdash Y:\tau_y \quad \mathcal{A}\vdash e_1:\tau_y \quad \mathcal{A}_Y\vdash e_2:\tau_x}{\mathcal{A}\vdash let_p\ Y=e_1\ in\ e_2:\tau_x}[\mathbf{LET}_p] \quad \mathcal{A}\oplus\{X:\tau_x\}\vdash X:\tau_x \quad \mathcal{A}\oplus\{X:Gen(\tau_x,\mathcal{A})\}\vdash e_3:\tau}{\mathcal{A}\vdash let_p\ X=(let_p\ Y=e_1\ in\ e_2)\ in\ e_3:\tau}$$

With this derivations as hypothesis we can build a type derivation of the new expression

$$[\mathbf{LET}_p]\dfrac{\dfrac{\mathcal{A}_Y \oplus \{X : \tau_x\} \vdash X : \tau_x \quad \mathcal{A}_Y \vdash e_2 : \tau_x \quad \mathcal{A}_Y \oplus \{X : Gen(\tau_x, \mathcal{A}_Y)\} \vdash e_3 : \tau}{\mathcal{A}_Y \vdash let_p\ X = e_2\ in\ e_3 : \tau} [\mathbf{LET}_p] \quad \mathcal{A} \oplus \{Y : \tau_y\} \vdash Y : \tau_y \quad \mathcal{A} \vdash e_1 : \tau_y}{\mathcal{A} \vdash let_p\ Y = e_1\ in\ (let_p\ X = e_2\ in\ e_3) : \tau}$$

$\mathcal{A} \oplus \{Y : \tau_y\} \vdash Y : \tau_y$, $\mathcal{A} \vdash e_1 : \tau_y$ and $\mathcal{A}_Y \vdash e_2 : \tau_x$ are the same derivations that appear in the original type derivation; and $\mathcal{A}_Y \oplus \{X : \tau_x\} \vdash X : \tau_x$ holds trivially applying the **[ID]** rule. But the derivation $\mathcal{A}_Y \oplus \{X : Gen(\tau_x, \mathcal{A}_Y)\} \vdash e_3 : \tau$ has to be proven. As before, since $Y \notin FV(e_3)$ by Theorem 1-b we can add an assumption over the Y and the derivation $(\mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A})\}) \oplus \{Y : Gen(\tau_y, \mathcal{A})\} \vdash e_3 : \tau$ will remain valid. Because $X \neq Y$ then by Observation 3 $(\mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A})\}) \oplus \{Y : Gen(\tau_y, \mathcal{A})\}$ is the same as $(\mathcal{A} \oplus \{Y : Gen(\tau_y, \mathcal{A})\}) \oplus \{X : Gen(\tau_x, \mathcal{A})\}$, and the derivation $(\mathcal{A} \oplus \{Y : Gen(\tau_y, \mathcal{A})\}) \oplus \{X : Gen(\tau_x, \mathcal{A})\} \vdash e_3 : \tau$ will be correct. Clearly $Gen(\tau_x, \mathcal{A}_Y)$ is not equal to $Gen(\tau_x, \mathcal{A})$ because a previous assumption for Y can be shadowed so that some free type variables in $\mathcal{A}$ are not in $\mathcal{A}_Y$. In the generalization step this means that some variables can be generalized in $Gen(\tau_x, \mathcal{A}_Y)$ but not in $Gen(\tau_x, \mathcal{A})$. The other case never happens because adding $\{Y : Gen(\tau_y, \mathcal{A})\}$ to $\mathcal{A}$ never adds free type variables: if some type variable in τ_y is not in $FTV(\mathcal{A})$ then it will be generalized and will not be in $FTV(\mathcal{A}_Y)$ either. Therefore $Gen(\tau_x, \mathcal{A}_Y) \succ Gen(\tau_x, \mathcal{A})$, and by Theorem 1-d the derivation $(\mathcal{A} \oplus \{Y : Gen(\tau_y, \mathcal{A})\}) \oplus \{X : Gen(\tau_x, \mathcal{A}_Y)\} \vdash e_3 : \tau$ is valid.

- $\mathcal{P} \vdash let_p\ X = (let_m\ Y = e_1\ in\ e_2)\ in\ e_3 \rightarrow^l let_p\ Y = e_1\ in\ (let_p\ X = e_2\ in\ e_3)$.
  The type derivation of the original expression is:

$$[\mathbf{LET}_p]\dfrac{\dfrac{\mathcal{A} \oplus \{Y : \tau_y\} \vdash Y : \tau_y \quad \mathcal{A} \vdash e_1 : \tau_y \quad \mathcal{A} \oplus \{Y : \tau_y\} \vdash e_2 : \tau_x}{\mathcal{A} \vdash let_m\ Y = e_1\ in\ e_2 : \tau_x} [\mathbf{LET}_m] \quad \mathcal{A} \oplus \{X : \tau_x\} \vdash X : \tau_x \quad \mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A})\} \vdash e_3 : \tau}{\mathcal{A} \vdash let_p\ X = (let_m\ Y = e_1\ in\ e_2)\ in\ e_3 : \tau}$$

  and we want to build one of the form (being $\mathcal{A}_Y \equiv \mathcal{A} \oplus \{Y : Gen(\tau_y, \mathcal{A})\}$):

$$[\mathbf{LET}_p]\dfrac{\dfrac{\mathcal{A}_Y \oplus \{X : \tau_x\} \vdash X : \tau_x \quad \mathcal{A}_Y \vdash e_2 : \tau_x \quad \mathcal{A}_Y \oplus \{X : Gen(\tau_x, \mathcal{A}_Y)\} \vdash e_3 : \tau}{\mathcal{A}_Y \vdash let_p\ X = e_2\ in\ e_3 : \tau} [\mathbf{LET}_p] \quad \mathcal{A} \oplus \{Y : \tau_y\} \vdash Y : \tau_y \quad \mathcal{A} \vdash e_1 : \tau_y}{\mathcal{A} \vdash let_p\ Y = e_1\ in\ (let_p\ X = e_2\ in\ e_3) : \tau}$$

The derivations $\mathcal{A} \oplus \{Y : \tau_y\} \vdash Y : \tau_y$ and $\mathcal{A} \vdash e_1 : \tau_y$ come from the original derivation; and $\mathcal{A}_Y \oplus \{X : \tau_x\} \vdash X : \tau_x$ is the trivial application of the [ID] rule. From the original derivation we have $\mathcal{A} \oplus \{Y : \tau_y\} \vdash e_2 : \tau_x$. It is easy to see that $Gen(\tau_y, \mathcal{A}) \succ \tau_y$, so by Theorem 1-d $\mathcal{A}_Y \vdash e_2 : \tau_x$. We also have from the original derivation that $\mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A})\} \vdash e_3 : \tau$. We know that $Y \notin FV(e_3)$, so by Theorem 1-b we can add an assumption over that variable and the derivation $(\mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A})\}) \oplus \{Y : Gen(\tau_y, \mathcal{A})\} \vdash e_3 : \tau$ will be valid. X is different from Y, so according to Observation 3 the set of assumptions $(\mathcal{A} \oplus \{X : Gen(\tau_x, \mathcal{A})\}) \oplus \{Y : Gen(\tau_y, \mathcal{A})\}$ is the same as $(\mathcal{A} \oplus \{Y : Gen(\tau_y, \mathcal{A})\}) \oplus \{X : Gen(\tau_x, \mathcal{A})\}$. By the same reasons given in the previous case $Gen(\tau_x, \mathcal{A}_Y) \succ Gen(\tau_x, \mathcal{A})$, so by Theorem 1-d the derivation $\mathcal{A}_Y \oplus \{X : Gen(\tau_x, \mathcal{A}_Y)\} \vdash e_3 : \tau$ will be valid.

– **(LetAp)** We will distinguish between the different let expressions.

let_m) The rewriting step is $\mathcal{P} \vdash (let_m\ X = e_1\ in\ e_2)e_3 \rightarrow^l let_m\ X = e_1\ in\ e_2e_3$. The type derivation of $(let_m\ X = e_1\ in\ e_2)e_3$ is:

$$[\mathbf{APP}]\dfrac{[\mathbf{LET}_m]\ \dfrac{\begin{array}{c}\mathcal{A} \oplus \{X : \tau_t\} \vdash X : \tau_t \\ \mathcal{A} \vdash e_1 : \tau_t \\ \mathcal{A} \oplus \{X : \tau_t\} \vdash e_2 : \tau_1 \rightarrow \tau\end{array}}{\mathcal{A} \vdash let_m\ X = e_1\ in\ e_2 : \tau_1 \rightarrow \tau} \qquad \mathcal{A} \vdash e_3 : \tau_1}{\mathcal{A} \vdash (let_m\ X = e_1\ in\ e_2)e_3 : \tau}$$

We want to construct a type derivation of the form:

$$[\mathbf{LET}_m]\dfrac{[\mathbf{APP}]\ \dfrac{\begin{array}{c}\mathcal{A} \oplus \{X : \tau_t\} \vdash e_2 : \tau_1 \rightarrow \tau \\ \mathcal{A} \oplus \{X : \tau_t\} \vdash e_3 : \tau_1\end{array}}{\mathcal{A} \oplus \{X : \tau_t\} \vdash e_2e_3 : \tau} \qquad \begin{array}{c}\mathcal{A} \oplus \{X : \tau_t\} \vdash X : \tau_t \\ \mathcal{A} \vdash e_1 : \tau_t\end{array}}{\mathcal{A} \vdash let_m\ X = e_1\ in\ e_2e_3 : \tau}$$

All the derivations appear in the original derivation, except $\mathcal{A} \oplus \{X : \tau_t\} \vdash e_3 : \tau_1$. Because we are using **(LetAp)**, we are sure that X does not appear in $FV(e_3)$. From the original derivation we have that $\mathcal{A} \vdash e_3 : \tau_1$, and by Theorem 1-b we can add an assumption over the variable X and obtain the derivation $\mathcal{A} \oplus \{X : \tau_t\} \vdash e_3 : \tau_1$.

let_p) Similar to the let_m) case.

– **(Contx)** We have a derivation $\mathcal{A} \vdash \mathcal{C}[e] : \tau$, so according to the Observation 7 in that derivation will appear a derivation a) $\mathcal{A} \oplus \mathcal{A}' \vdash e : \tau'$, being $\mathcal{A}'$ a set of assumptions over variables. If we apply the rule **(Contx)** to reduce an expression $\mathcal{C}[e]$ is because we reduce the expression e using any of the other rules of the let-rewriting relation b) $\mathcal{P} \vdash e \rightarrow^l e'$. We also know by Observation 8 that c) $wt_{\mathcal{A} \oplus \mathcal{A}'}(\mathcal{P})$. With a), b) and c) the Induction Hypothesis states that $\mathcal{A} \oplus \mathcal{A}' \vdash e' : \tau'$, and by Lemma 5 then $\mathcal{A} \vdash \mathcal{C}[e'] : \tau$.

□

Theorem 4 (Soundness of $\Vdash$ wrt $\vdash$)

1) $\mathcal{A} \Vdash e : \tau|\pi \Longrightarrow \mathcal{A}\pi \vdash e : \tau$

Proof.
We proceed by induction over the size of the type inference $\mathcal{A} \Vdash e : \tau|\pi$.
Base Case

- **[iID]** We have a type inference of the form:

$$[\mathbf{iID}]\frac{}{\mathcal{A} \Vdash g : \tau|id}$$

where $\mathcal{A}(g) = \sigma$ and τ is a variant of σ. It is clear that if τ is a variant of σ it is also a generic instance of σ, and $\mathcal{A}\ id \equiv \mathcal{A}$ so the following type derivation is valid:

$$[\mathbf{ID}]\frac{}{\mathcal{A} \vdash g : \tau}$$

Induction Step

- **[iAPP]** The type inference is:

$$[\mathbf{iAPP}]\frac{\mathcal{A} \Vdash e_1 : \tau_1|\pi_1 \qquad \mathcal{A}\pi_1 \Vdash e_2 : \tau_2|\pi_2}{\mathcal{A} \Vdash e_1 e_2 : \alpha\pi|\pi_1\pi_2\pi}$$

where $\pi = mgu(\tau_1\pi_2, \tau_2 \to \alpha)$, being α a fresh type variable. By the Induction Hypothesis we have that $\mathcal{A}\pi_1 \vdash e_1 : \tau_1$ and $\mathcal{A}\pi_1\pi_2 \vdash e_2 : \tau_2$. We can apply Theorem 1-a to both derivations and obtain $\mathcal{A}\pi_1\pi_2\pi \vdash e_1 : \tau_1\pi_2\pi$ and $\mathcal{A}\pi_1\pi_2\pi \vdash e_2 : \tau_2\pi$. Since we know that $\tau_1\pi_2\pi = (\tau_2 \to \alpha)\pi = \tau_2\pi \to \alpha\pi$ then we can construct the type derivation:

$$[\mathbf{APP}]\frac{\mathcal{A}\pi_1\pi_2\pi \vdash e_1 : \tau_2\pi \to \alpha\pi \qquad \mathcal{A}\pi_1\pi_2\pi \vdash e_2 : \tau_2\pi}{\mathcal{A}\pi_1\pi_2\pi \vdash e_1 e_2 : \alpha\pi}$$

- **[iΛ]** The type inference will be of the form:

$$[\mathbf{i}\Lambda]\frac{\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_t|\pi_t \qquad (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t \Vdash e : \tau|\pi}{\mathcal{A} \Vdash \lambda t.e : \tau_t\pi \to \tau|\pi_t\pi}$$

where $\overline{\alpha_i}$ are fresh type variables. By the Induction Hypothesis we have that $\mathcal{A}\pi_t \oplus \{\overline{X_i : \alpha_i\pi_t}\} \vdash t : \tau_t$ and $\mathcal{A}\pi_t\pi \oplus \{\overline{X_i : \alpha_i\pi_t\pi}\} \vdash e : \tau$. We can apply Theorem 1-a to the first derivation and obtain $\mathcal{A}\pi_t\pi \oplus \{\overline{X_i : \alpha_i\pi_t\pi}\} \vdash t : \tau_t\pi$. Therefore the following type derivation is correct:

$$[\Lambda]\frac{\mathcal{A}\pi_t\pi \oplus \{\overline{X_i : \alpha_i\pi_t\pi}\} \vdash t : \tau_t\pi \qquad \mathcal{A}\pi_t\pi \oplus \{\overline{X_i : \alpha_i\pi_t\pi}\} \vdash e : \tau}{\mathcal{A}\pi_t\pi \vdash \lambda t.e : \tau_t\pi \to \tau}$$

- **[iLET$_m$]** In this case the type inference will be:

$$[\mathbf{iLET}_m]\frac{\begin{array}{c}\mathcal{A}\oplus\{\overline{X_i:\alpha_i}\}\Vdash t:\tau_t|\pi_t\\ \mathcal{A}\pi_t\Vdash e:\tau_1|\pi_1\\ (\mathcal{A}\oplus\{\overline{X_i:\alpha_i}\})\pi_t\pi_1\pi\Vdash e_2:\tau_2|\pi_2\end{array}}{\mathcal{A}\Vdash let_m\ t=e_1\ in\ e_2:\tau_2|\pi_t\pi_1\pi\pi_2}$$

where $\overline{\alpha_i}$ are fresh type variables and $\pi = mgu(\tau_t\pi_1, \tau_1)$. By the Induction Hypothesis we have that $\mathcal{A}\pi_t \oplus \{\overline{X_i : \alpha_i\pi_t}\} \vdash t : \tau_t$, $\mathcal{A}\pi_t\pi_1 \vdash e : \tau_1$ and $\mathcal{A}\pi_t\pi_1\pi\pi_2 \oplus \{\overline{X_i : \alpha_i\pi_t\pi_1\pi\pi_2}\} \vdash e_2 : \tau_2$. We can apply Theorem 1-a to the first two derivations and obtain $\mathcal{A}\pi_t\pi_1\pi\pi_2 \oplus \{\overline{X_i : \alpha_i\pi_t\pi_1\pi\pi_2}\} \vdash t : \tau_t\pi_1\pi\pi_2$ and $\mathcal{A}\pi_t\pi_1\pi\pi_2 \vdash e : \tau_1\pi\pi_2$. Finally, as $\tau_t\pi_1\pi = \tau_1\pi$ then we can build a type derivation of the form:

$$[\mathbf{LET}_m]\frac{\begin{array}{c}\mathcal{A}\pi_t\pi_1\pi\pi_2\oplus\{\overline{X_i:\alpha_i\pi_t\pi_1\pi\pi_2}\}\vdash t:\tau_1\pi\pi_2\\ \mathcal{A}\pi_t\pi_1\pi\pi_2\vdash e:\tau_1\pi\pi_2\\ \mathcal{A}\pi_t\pi_1\pi\pi_2\oplus\{\overline{X_i:\alpha_i\pi_t\pi_1\pi\pi_2}\}\vdash e_2:\tau_2\end{array}}{\mathcal{A}\pi_t\pi_1\pi\pi_2\vdash let_m\ t=e_1\ in\ e_2:\tau_2}$$

– $[\mathbf{iLET}^X_{pm}]$ The inference will be:

$$[\mathbf{LET}^X_{pm}]\frac{\mathcal{A}\Vdash e_1:\tau_1|\pi_1 \qquad \mathcal{A}\pi_1\oplus\{X:Gen(\tau_1,\mathcal{A}\pi_1)\}\Vdash e_2:\tau_2|\pi_2}{\mathcal{A}\Vdash let_{pm}\ X=e_1\ in\ e_2:\tau_2|\pi_1\pi_2}$$

By the Induction Hypothesis we have the type derivations $\mathcal{A}\pi_1 \vdash e_1 : \tau_1$ and $\mathcal{A}\pi_1\pi_2 \oplus \{X : Gen(\tau_1, \mathcal{A}\pi_1)\pi_2\} \vdash e_2 : \tau_2$. We can construct a type substitution $\pi \in \mathcal{TS}ubst$ such that maps the type variables in $FTV(\tau_1) \setminus FTV(\mathcal{A}\pi_1)$ to fresh variables. Then it is clear that $Gen(\tau_1, \mathcal{A}\pi_1) = Gen(\tau_1\pi, \mathcal{A}\pi_1)$. On the other hand, all the variables in $\tau_1\pi$ which are not in $FTV(\mathcal{A}\pi_1)$ are fresh so they do not appear in π_2, and by Lemma 7 $Gen(\tau_1\pi, \mathcal{A}\pi_1)\pi_2 = Gen(\tau_1\pi\pi_2, \mathcal{A}\pi_1\pi_2)$. Therefore the type derivation

$$\mathcal{A}\pi_1\pi_2 \oplus \{X : Gen(\tau_1\pi\pi_2, \mathcal{A}\pi_1\pi_2)\} \vdash e_2 : \tau_2$$

is correct. By Theorem 1-a we obtain $\mathcal{A}\pi_1\pi\pi_2 \vdash e_1 : \tau_1\pi\pi_2$, and as $Dom(\pi) \cap FTV(\mathcal{A}\pi_1) = \emptyset$ then $\mathcal{A}\pi_1\pi_2 \vdash e_1 : \tau_1\pi\pi_2$.
Finally with these derivations we can build the type derivation we intended:

$$[\mathbf{LET}^X_{pm}]\frac{\mathcal{A}\pi_1\pi_2\vdash e_1:\tau_1\pi\pi_2 \qquad \mathcal{A}\pi_1\pi_2\oplus\{X:Gen(\tau_1\pi\pi_2,\mathcal{A}\pi_1\pi_2)\}\vdash e_2:\tau_2}{\mathcal{A}\pi_1\pi_2\vdash let_{pm}\ X=e_1\ in\ e_2:\tau_2}$$

– $[\mathbf{iLET}^h_{pm}]$ This case is similar to the $[\mathbf{LET}_m]$ case.
– $[\mathbf{iLET}_p]$ In this case we have an inference of the form:

$$[\mathbf{iLET}_p]\frac{\begin{array}{c}\mathcal{A}\oplus\{\overline{X_i:\alpha_i}\}\Vdash t:\tau_t|\pi_t\\ \mathcal{A}\pi_t\Vdash e_1:\tau_1|\pi_1\\ \mathcal{A}\pi_t\pi_1\pi\oplus\{\overline{X_i:Gen(\alpha_i\pi_t\pi_1\pi,\mathcal{A}\pi_t\pi_1\pi)}\}\Vdash e_2:\tau_2|\pi_2\end{array}}{\mathcal{A}\Vdash let_p\ t=e_1\ in\ e_2:\tau_2|\pi_t\pi_1\pi\pi_2}$$

where $\pi = mgu(\tau_t\pi_1, \tau_1)$. By the Induction Hypothesis we have $\mathcal{A}\pi_t\pi_1\pi\pi_2 \oplus \{\overline{X_i : Gen(\alpha_i\pi_t\pi_1\pi, \mathcal{A}\pi_t\pi_1\pi)\pi_2}\} \vdash e_2 : \tau_2$ and $\mathcal{A}\pi_t \oplus \{\overline{X_i : \alpha_i\pi_t}\} \vdash t : \tau_t$, $\mathcal{A}\pi_t\pi_1 \vdash e_1 : \tau_1$. Let be $\overline{\beta_i}$ the type variables in all the types $\alpha_i\pi_t\pi_1\pi$ which do

not appear in $\mathcal{A}\pi_t\pi_1\pi$. We can create a type substitution π' from $\overline{\beta_i}$ to fresh variables. It is clear that $Gen(\alpha_i\pi_t\pi_1\pi, \mathcal{A}\pi_t\pi_1\pi) = Gen(\alpha_i\pi_t\pi_1\pi\pi', \mathcal{A}\pi_t\pi_1\pi)$, as π' only substitutes the variables that will be generalized by fresh ones which will also be generalized, so it is a renaming of the bounded variables (Observation 1). Therefore the derivation

$$\mathcal{A}\pi_t\pi_1\pi\pi_2 \oplus \{\overline{X_i : Gen(\alpha_i\pi_t\pi_1\pi\pi', \mathcal{A}\pi_t\pi_1\pi)\pi_2}\} \vdash e_2 : \tau_2$$

is also valid. Applying the Theorem 1-a to the first two derivations we obtain $\mathcal{A}\pi_t\pi_1\pi\pi'\pi_2 \oplus \{\overline{X_i : \alpha_i\pi_t\pi_1\pi\pi'\pi_2}\} \vdash t : \tau_t\pi_1\pi\pi'\pi_2$ and $\mathcal{A}\pi_t\pi_1\pi\pi'\pi_2 \vdash e_1 : \tau_1\pi\pi'\pi_2$. By construction, no variable in $Dom(\pi')$ or $Rng(\pi')$ is in $FTV(\mathcal{A}\pi_t\pi_1\pi)$, so $\mathcal{A}\pi_t\pi_1\pi\pi'\pi_2 = \mathcal{A}\pi_t\pi_1\pi\pi_2$. By Lemma 7 we know that $Gen(\alpha_i\pi_t\pi_1\pi\pi', \mathcal{A}\pi_t\pi_1\pi)\pi_2 = Gen(\alpha_i\pi_t\pi_1\pi\pi'\pi_2, \mathcal{A}\pi_t\pi_1\pi\pi_2)$, so the derivation $\mathcal{A}\pi_t\pi_1\pi\pi_2 \oplus \{\overline{X_i : Gen(\alpha_i\pi_t\pi_1\pi\pi'\pi_2, \mathcal{A}\pi_t\pi_1\pi\pi_2)}\} \vdash e_2 : \tau_2$ is correct.
With this derivations as premises we can build the expected one:

$$[\mathbf{LET}_p]\frac{\mathcal{A}\pi_t\pi_1\pi\pi_2 \oplus \{\overline{X_i : \alpha_i\pi_t\pi_1\pi\pi'\pi_2}\} \vdash t : \tau_1\pi\pi'\pi_2 \quad \mathcal{A}\pi_t\pi_1\pi\pi_2 \vdash e_1 : \tau_1\pi\pi'\pi_2 \quad \mathcal{A}\pi_t\pi_1\pi\pi_2 \oplus \{\overline{X_i : Gen(\alpha_i\pi_t\pi_1\pi\pi'\pi_2, \mathcal{A}\pi_t\pi_1\pi\pi_2)}\} \vdash e_2 : \tau_2}{\mathcal{A}\pi_t\pi_1\pi\pi_2 \vdash let_p\ t = e_1\ in\ e_2 : \tau_2}$$

(remembering that $\tau_t\pi_1\pi = \tau_1\pi$ because of π is a mgu).

□

---

2) $\mathcal{A} \Vdash^\bullet e : \tau|\pi \Longrightarrow \mathcal{A}\pi \vdash^\bullet e : \tau$

By definition of $\Vdash^\bullet$ we have that $\mathcal{A} \Vdash e : \tau$ and $critVar_{\mathcal{A}\pi}(e)$. Applying the soundness of $\Vdash$ (Theorem 4) we have that $\mathcal{A}\pi \vdash e : \tau$. Since $\mathcal{A}\pi \vdash e : \tau$ and $critVar_{\mathcal{A}\pi}(e)$, then by definition of $\vdash^\bullet$ we have $\mathcal{A}\pi \vdash^\bullet e : \tau$.

□

Theorem 5 (Completeness of $\Vdash$ wrt $\vdash$).
$\mathcal{A}\pi' \vdash e : \tau' \Longrightarrow \exists\tau, \pi, \pi''.\ \mathcal{A} \Vdash e : \tau|\pi \wedge \mathcal{A}\pi\pi'' = \mathcal{A}\pi' \wedge \tau\pi'' = \tau'$.

Proof.
This proof is based on the proof of completeness of algorithm $\mathcal{W}$ in [12]. We proceed by induction over the size of the type derivation.
Base Case

- **[ID]** In this case we have a type derivation:

$$[\mathbf{ID}]\frac{}{\mathcal{A}\pi' \vdash s : \tau'}$$

if $\mathcal{A}\pi'(s) = \sigma$ and $\sigma \succ \tau'$. Let's suppose that $\mathcal{A}(s) = \forall\overline{\alpha_i}.\tau''$ (with $\overline{\alpha}$ fresh variables), then $\sigma \equiv (\forall\overline{\alpha_i}.\tau'')\pi' = \forall\overline{\alpha_i}.(\tau''\pi')$. Since $\sigma \succ \tau'$ then there exists a type substitution $[\overline{\alpha_i/\tau_i}]$ such that $\tau' = (\tau''\pi')[\overline{\alpha_i/\tau_i}]$.
Let $\overline{\beta_i}$ be fresh variables. As $\tau''[\overline{\alpha_i/\beta_i}]$ is a variant of $\forall\overline{\alpha_i}.\tau''$ then the following type inference is correct:

$$[\mathbf{iID}]\frac{}{\mathcal{A} \Vdash s : \tau''[\overline{\alpha_i/\beta_i}]|id}$$

There is also a type substitution $\pi'' \equiv \pi'[\overline{\beta_i/\tau_i}]$ such that $\tau''[\overline{\alpha_i/\beta_i}]\pi'' = \tau''[\overline{\alpha_i/\beta_i}]\pi'[\overline{\beta_i/\tau_i}] = (\tau''\pi')[\overline{\alpha_i/\beta_i}][\overline{\beta_i/\tau_i}] = (\tau''\pi')[\overline{\alpha_i/\tau_i}] = \tau'$. Finally, it is clear that $\mathcal{A}id\pi'' = \mathcal{A}id\pi'[\overline{\beta_i/\tau_i}] = \mathcal{A}\pi'[\overline{\beta_i/\tau_i}] = \mathcal{A}\pi'$ because $\overline{\beta_i}$ are fresh and cannot occur in $FTV(\mathcal{A}\pi')$.

Induction Step

– **[APP]** The type derivation will be:

$$[\mathbf{APP}]\frac{\mathcal{A}\pi' \vdash e_1 : \tau_1' \to \tau' \quad \mathcal{A}\pi' \vdash e_2 : \tau_1'}{\mathcal{A}\pi' \vdash e_1 e_2 : \tau'}$$

By the Induction Hypothesis we know that $\mathcal{A} \Vdash e_1 : \tau_1|\pi_1$ and there is a type substitution π_1'' such that $\tau_1\pi_1'' = \tau_1' \to \tau'$ and $\mathcal{A}\pi' = \mathcal{A}\pi_1\pi_1''$. Since $\mathcal{A}\pi' = \mathcal{A}\pi_1\pi_1''$ then the derivation $(\mathcal{A}\pi_1)\pi_1'' \vdash e_2 : \tau_1'$ is correct, and again by the Induction Hypothesis we know that $\mathcal{A}\pi_1 \Vdash e_2 : \tau_2|\pi_2$ and that there exists a type substitution π_2'' such that $\tau_2\pi_2'' = \tau_1'$ and $\mathcal{A}\pi_1\pi_1'' = \mathcal{A}\pi_1\pi_2\pi_2''$. We can assume that π_2'' is minimal, so $Dom(\pi_2'') \subseteq FTV(\tau_2) \cup FTV(\mathcal{A}\pi_1\pi_2)$. In order to prove that the existence of a type inference $\mathcal{A} \Vdash e_1 : \alpha\pi|\pi_1\pi_2\pi$ we need to prove that there exists a most general unifier for $\tau_1\pi_2$ and $\tau_2 \to \alpha$ (being α a fresh variable). For that, we will construct a type substitution π_u which will unify these two types. We know that $\mathcal{A}\pi_1\pi_1'' = \mathcal{A}\pi_1\pi_2\pi_2''$, so for all the variables which are free in $\mathcal{A}\pi_1$ then $\pi_1'' = \pi_2\pi_2''$. Let α a fresh type variable, $B = Dom(\pi_1'') \setminus FTV(\mathcal{A}\pi_1)$ and $\pi_u \equiv \pi_2'' + \pi_1''|_B + [\alpha/\tau']$. π_u is well defined because the domains of the three substitutions are disjoints. According to Observation 6, the variables in $FTV(\tau_2)$, $Dom(\pi_2)$ or $Rng(\pi_2)$ which are not in $FTV(\mathcal{A}\pi_1)$ are fresh variables and cannot occur in B. Since the variables in B are neither in $FTV(\mathcal{A}\pi_1)$ nor in $Rng(\pi_2)$ then they do not appear in $FTV(\mathcal{A}\pi_1\pi_2)$ either; and as π_2'' is minimal then no variable in B could occur in $Dom(\pi_2'')$. Besides α is fresh, and it can occur neither in π_2'' nor in $\pi_1''|_B$. Applying π_u to $\tau_2 \to \alpha$ we obtain $(\tau_2 \to \alpha)\pi_u = \tau_2\pi_u \to \alpha\pi_u = \tau_2\pi_2'' \to \alpha[\alpha/\tau'] = \tau_1' \to \tau'$. On the other hand, $\tau_1\pi_2\pi_u = \tau_1' \to \tau'$ because if a type variable of τ_1 is in $\mathcal{A}\pi_1$ then $\tau_1\pi_2\pi_u = \tau_1\pi_2\pi_2'' = \tau_1\pi_1'' = \tau_1' \to \tau'$, and if not it will be in B and π_2 will not affect it, so $\tau_1\pi_2\pi_u = \tau_1\pi_u = \tau_1\pi_1''|_B = \tau_1' \to \tau'$. Since π_u is an unifier, then there will exists a most general unifier π of $\tau_1\pi_2$ and $\tau_2 \to \alpha$ [19]. Therefore the following type inference is correct:

$$[\mathbf{iAPP}]\frac{\mathcal{A} \Vdash e_1 : \tau_1|\pi_1 \quad \mathcal{A}\pi_1 \Vdash e_2 : \tau_2|\pi_2}{\mathcal{A} \vdash e_1 e_2 : \alpha\pi|\pi_1\pi_2\pi}$$

Now we have to prove that there exists a type substitution π'' such that $\alpha\pi\pi'' = \tau'$ and $\mathcal{A}\pi' = \mathcal{A}\pi_1\pi_2\pi\pi''$. This is easy defining π'' such that $\pi_u = \pi\pi''$ (which is well defined as π_u is an unifier and π is the most general unifier). Then it is clear that $\alpha\pi\pi'' = \alpha\pi_u = \alpha[\alpha/\tau'] = \tau'$ and $\mathcal{A}\pi' = \mathcal{A}\pi_1\pi_1'' = \mathcal{A}\pi_1\pi_2\pi_2'' = \mathcal{A}\pi_1\pi_2\pi_u = \mathcal{A}\pi_2\pi_2\pi\pi''$.

– [Λ] We assume that the variables $\overline{X_i}$ in the pattern t do not appear in $\mathcal{A}\pi'$ (nor in $\mathcal{A}$). In this case the type derivation is:

$$[\Lambda]\frac{\mathcal{A}\pi' \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau'_t \quad \mathcal{A}\pi' \oplus \{\overline{X_i : \tau_i}\} \vdash e : \tau'}{\mathcal{A}\pi' \vdash \lambda t.e : \tau'_t \to \tau'}$$

Let $\overline{\alpha_i}$ be fresh type variables and $\pi_g \equiv [\overline{\alpha_i/\tau_i}]$. Then the first derivation is equal to $(\mathcal{A} \oplus \{X_i : \alpha_i\})\pi'\pi_g \vdash t : \tau'_t$. By the Induction Hypothesis we know that $\mathcal{A}\oplus\{\overline{X_i : \alpha_i}\} \Vdash t : \tau_t|\pi_t$ and that exists a type substitution π''_t such that $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi'\pi_g = (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi''_t$ and $\tau_t\pi''_t = \tau'_t$. Because the data variables $\overline{X_i}$ do not appear in $\mathcal{A}$, then it is true that $\mathcal{A}\pi'\pi_g = \mathcal{A}\pi' = \mathcal{A}\pi_t\pi''_t$ and for every type variable $\alpha_i\pi'\pi_g = \alpha_i\pi_g = \tau_i = \alpha\pi_t\pi''_t$.
Using these equalities we can write $\mathcal{A}\pi'\oplus\{\overline{X_i : \tau_i}\}$ as $\mathcal{A}\pi_t\pi''_t\oplus\{\overline{X_i : \alpha_i\pi_t\pi''_t}\}$, that is the same as $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi''_t$. Then, the second derivation is equal to $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi''_t \vdash e : \tau'$, and by the Induction Hypothesis $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t \Vdash e : \tau_e|\pi_e$ and there exists a type substitution π''_e such that $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi''_t = (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_e\pi''_e$ and $\tau_e\pi''_e = \tau'$. As before, it is also true that $\mathcal{A}\pi_t\pi''_t = \mathcal{A}\pi_t\pi_e\pi''_e$ and for every type variable $\alpha_i\pi_t\pi''_t = \alpha\pi_t\pi_e\pi''_e$. We can assume that π''_e is minimal, so $Dom(\pi''_e) \subseteq FTV(\tau_e) \cup FTV((\mathcal{A} \cup \{\overline{X_i : \alpha_i}\})\pi_t\pi_e)$. Therefore the type inference for the lambda expression exists and have the form:

$$[\mathbf{i}\Lambda]\frac{\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_t|\pi_t \quad (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t \Vdash e : \tau_e|\pi_e}{\mathcal{A} \Vdash \lambda t.e : \tau_t\pi_e \to \tau_e|\pi_t\pi_e}$$

Now we have to prove that there exists a type substitution π'' such that $\mathcal{A}\pi' = \mathcal{A}\pi_t\pi_e\pi''$ and $(\tau_t\pi_e \to \tau_e)\pi'' = \tau'_t \to \tau'$. Let be $B \equiv Dom(\pi''_t) \setminus FTV((\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t)$ and $\pi'' \equiv \pi''_t|_B + \pi''_e$, which is well defined because the domains are disjoints. According to Observation 6, the variables which are not in $FTV((\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t)$ and appear in $FTV(\tau_e)$, $Dom(\pi_e)$ or in $Rng(\pi_e)$ are fresh, so they cannot be in B. As these variables do not appear in $Rng(\pi_e)$ then they do not appear in $FTV((\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_e)$; so the variables in B are not in $Dom(\pi''_e)$ and the domains of π''_e and $\pi''_t|_B$ are disjoints.
It is clear that $\mathcal{A}\pi' = \mathcal{A}\pi_t\pi''_t = \mathcal{A}\pi_t\pi_e\pi''_e = \mathcal{A}\pi_t\pi_e\pi''$ because π''_e is part of π''. To prove that $(\tau_t\pi_e \to \tau_e)\pi'' = \tau'_t \to \tau'$ we need to prove that $\tau_t\pi_e\pi'' = \tau'_t$ and $\tau_e\pi'' = \tau'$. The second part is straightforward because $\tau' = \tau_e\pi''_e = \tau_e\pi''$. To prove the first one we will distinguish over the type variables in τ_t. For all the type variables of τ_t which are in $(\mathcal{A}\oplus\{\overline{X_i : \alpha_i}\})\pi_t$ (i.e. they are not in B) we know that $\tau_t\pi_e\pi'' = \tau_t\pi_e\pi''_e = \tau_t\pi''_t = \tau'_t$ because $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi''_t = (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_e\pi''_e$. For the variables in τ_t which are in B the case is simpler because we know they do not appear in $Dom(\pi_e)$, therefore so $\tau_t\pi_e\pi'' = \tau_t\pi'' = \tau_t\pi''_t|_B = \tau'_t$.
– [$\mathbf{LET}_m$] We assume that the variables $\overline{X_i}$ of the pattern t are fresh and do not occur in $\mathcal{A}\pi'$ (nor in $\mathcal{A}$). Then the type derivation will be:

$$[\mathbf{LET}_m]\frac{\begin{array}{c}\mathcal{A}\pi' \oplus \{\overline{X_i : \tau_i}\} \vdash t : \tau'_t \\ \mathcal{A}\pi' \vdash e_1 : \tau'_t \\ \mathcal{A}\pi' \oplus \{\overline{X_i : \tau_i}\} \vdash e_2 : \tau'\end{array}}{\mathcal{A}\pi' \vdash let_m\ t = e_1\ in\ e_2 : \tau'}$$

Let $\overline{\alpha_i}$ be fresh type variables, and $\pi_g \equiv [\overline{\alpha_i/\tau_i}]$. Since $\overline{\alpha_i}$ are fresh it is clear that $\mathcal{A}\pi'\pi_g = \mathcal{A}\pi'$ and $\alpha_i\pi'\pi_g = \alpha_i\pi_g = \tau_i$ for every type variable α_i. Then we can write the first derivation as $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi'\pi_g \vdash t : \tau'_t$ and by the Induction Hypothesis $\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\} \Vdash t : \tau_t|\pi_t$ and there is a type substitution π''_t such that $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi'\pi_g = (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi''_t$ and $\tau_t\pi''_t = \tau'_t$. Since the data variables $\overline{X_i}$ do not appear in $\mathcal{A}\pi'$ then $\mathcal{A}\pi' = \mathcal{A}\pi'\pi_g = \mathcal{A}\pi_t\pi''_t$ and for every type variable $\alpha_i\pi'\pi_g = \alpha_i\pi_g = \tau_i = \alpha_i\pi_t\pi''_t$. Since $\mathcal{A}\pi' = \mathcal{A}\pi_t\pi''_t$ then we can write the second derivation as $\mathcal{A}\pi_t\pi''_t \vdash e_1 : \tau'_t$, and by the Induction Hypothesis $\mathcal{A}\pi_t \Vdash e_1 : \tau_1|\pi_1$ and there exists a type substitution π''_1 such that $\mathcal{A}\pi_t\pi''_t = \mathcal{A}\pi_t\pi_1\pi''_1$ and $\tau_1\pi''_1 = \tau'_t$. We can assume that π''_1 is minimal, so $Dom(\pi''_1) \subseteq FTV(\tau_1) \cup FTV(\mathcal{A}\pi_t\pi_1)$. Now we have to prove that $\tau_t\pi_1$ and τ_1 are unifiable, so there exists a most general unifier [19]. We define $B \equiv FTV(\pi''_t) \setminus FTV(\mathcal{A}\pi_t)$ and $\pi_u \equiv \pi''_1 + \pi''_t|_B$, which is well defined because the domains of the two components are disjoints. According to Observation 6, the variables of $FTV(\tau_1)$, $Dom(\pi_1)$ or $Rng(\pi_1)$ which do not occur in $FTV(\mathcal{A}\pi_t)$ will be fresh variables, so they will not be any of the variables in B. As the variables in B occur neither in $FTV(\mathcal{A}\pi_t)$ nor in $Rng(\pi_1)$, then they do not appear in $\mathcal{A}\pi_1\pi_1$; and as π''_1 is minimal then no variable in B occurs in $Dom(\pi''_1)$.

π_u is an unifier of $\tau_t\pi_1$ and τ_1 because $\tau_t\pi_1\pi_u = \tau_1\pi_u = \tau'_t$. The first case is easy because $\tau_1\pi_u = \tau_1\pi''_1 = \tau'_t$. To prove the second we will distinguish over the type variables of τ_t. For the type variables of τ_t in $\mathcal{A}\pi_t$ (i.e. those which are not in B) we know that $\tau_t\pi_1\pi_u = \tau_t\pi_1\pi''_1 = \tau_t\pi''_t = \tau'_t$, and for the others (those in B) we know they are fresh and do not appear in π_1, so $\tau_t\pi_1\pi_u = \tau_t\pi_u = \tau_t\pi''_t|_B = \tau'_t$. Therefore there will exist a most general unifier π, and $\pi_u = \pi\pi_o$.

We also know that $\mathcal{A}\pi' = \mathcal{A}\pi_t\pi''_t = \mathcal{A}\pi_t\pi_1\pi''_1 = \mathcal{A}\pi_t\pi_1\pi_u = \mathcal{A}\pi_t\pi_1\pi\pi_o$ and for every type variable $\alpha_i\pi_t\pi_1\pi\pi_o = \tau_i$ (for the type variables of $\alpha_i\pi_t$ which are in $\mathcal{A}\pi_t$ then $\alpha_i\pi_t\pi_1\pi\pi_o = \alpha_i\pi_t\pi_1\pi_u = \alpha_i\pi_t\pi_1\pi''_1 = \alpha_i\pi_t\pi''_t = \tau_i$, and for the rest of the variables -those in B- then $\alpha_i\pi_t\pi_1\pi\pi_o = \alpha_i\pi_t\pi_1\pi_u = \alpha_i\pi_t\pi_u = \alpha_i\pi_t\pi''_t|_B = \tau_i$).

Then we can write the third derivation as $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_1\pi\pi_o \vdash e_2 : \tau'$, and by the Induction Hypothesis $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_1\pi \Vdash e_2 : \tau_2|\pi_2$ and there exists a type substitution π''_2 such that $\tau_2\pi''_2 = \tau'$ and $(\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_1\pi\pi_o = (\mathcal{A} \oplus \{\overline{X_i : \alpha_i}\})\pi_t\pi_1\pi\pi_2\pi''_2$. Since the variables $\overline{X_i}$ do not appear in $\mathcal{A}$, in particular it is true that $\mathcal{A}\pi_t\pi_1\pi\pi_o = \mathcal{A}\pi_t\pi_1\pi\pi_2\pi''_2$.

With these three type inferences we can build the type inference for the let expression:

$$[\mathbf{iLET}_m]\frac{\begin{array}{c}\mathcal{A}\oplus\{\overline{X_i:\alpha_i}\}\Vdash t:\tau_t|\pi_t\\ \mathcal{A}\pi_t\Vdash e_1:\tau_1|\pi_1\\ (\mathcal{A}\oplus\{\overline{X_i:\alpha_i}\})\pi_t\pi_1\pi\Vdash e_2:\tau_2|\pi_2\end{array}}{\mathcal{A}\Vdash let_m\ t=e_1\ in\ e_2:\tau_2|\pi_t\pi_1\pi\pi_2}$$

being $\pi = mgu(\tau_t\pi_1, \tau_1)$. To finish this case we only have to prove that there exists a type substitution π'' such that $\tau_2\pi'' = \tau'$ and $\mathcal{A}\pi' = \mathcal{A}\pi_t\pi_1\pi\pi_2\pi''$. This substitution π'' is π_2''.

– $[\mathbf{LET}^X_{pm}]$ We assume that X does not occur in $\mathcal{A}$. We have a type derivation:

$$[\mathbf{LET}^X_{pm}]\frac{\begin{array}{c}\mathcal{A}\pi'\vdash e_1:\tau_1'\\ \mathcal{A}\pi'\oplus\{X:Gen(\tau_1',\mathcal{A}\pi')\}\vdash e_2:\tau_2'\end{array}}{\mathcal{A}\pi'\vdash let_{pm}\ X=e_1\ in\ e_2:\tau_2'}$$

By the Induction Hypothesis we have that $\mathcal{A}\Vdash e_1:\tau_1|\pi_1$ and there exists a type substitution π_1'' such that $\mathcal{A}\pi' = \mathcal{A}\pi_1\pi_1''$ and $\tau_1\pi_1'' = \tau_1'$. $Gen(\tau_1', \mathcal{A}\pi') = Gen(\tau_1\pi_1'', \mathcal{A}\pi_1\pi_1'')$, so by Lemma 8 $Gen(\tau_1, \mathcal{A}\pi_1)\pi_1'' \succ Gen(\tau_1', \mathcal{A}\pi')$. Then by Theorem 1-d the type derivation $\mathcal{A}\pi'\oplus\{X : Gen(\tau_1, \mathcal{A}\pi_1)\pi_1''\}\vdash e_2 : \tau_2'$ is valid. We can write this derivation as $(\mathcal{A}\pi_1\oplus\{X : Gen(\tau_1, \mathcal{A}\pi_1)\})\pi_1''\vdash e_2 : \tau_2'$ and applying the Induction Hypothesis we obtain that $\mathcal{A}\pi_1\oplus\{X : Gen(\tau_1, \mathcal{A}\pi_1)\}\Vdash e_2 : \tau_2|\pi_2$ and there exists a type substitution π_2'' such that $\tau_2\pi_2'' = \tau_2'$ and $(\mathcal{A}\pi_1\oplus\{X : Gen(\tau_1, \mathcal{A}\pi_1)\})\pi_2\pi_2'' = (\mathcal{A}\pi_1\oplus\{X : Gen(\tau_1, \mathcal{A}\pi_1)\})\pi_1''$. Since X does not appear in $\mathcal{A}$ the last equality means that $\mathcal{A}\pi_1\pi_2\pi_2'' = \mathcal{A}\pi_1\pi_1''$ and $Gen(\tau_1, \mathcal{A}\pi_1)\pi_2\pi_2'' = Gen(\tau_1, \mathcal{A}\pi_1)\pi_1''$. With the previous type inferences we can construct a type inference for the whole expression:

$$[\mathbf{iLET}^X_{pm}]\frac{\begin{array}{c}\mathcal{A}\Vdash e_1:\tau_1|\pi_1\\ \mathcal{A}\pi_1\oplus\{X:Gen(\tau_1,\mathcal{A}\pi_1)\}\Vdash e_2:\tau_2|\pi_2\end{array}}{\mathcal{A}\Vdash let_{pm}\ X=e_1\ in\ e_2:\tau_2|\pi_1\pi_2}$$

In this case it is easy to see that there exists a type substitution (π_2'') such that $\tau_2\pi_2'' = \tau_2'$ and $\mathcal{A}\pi' = \mathcal{A}\pi_1\pi_1'' = \mathcal{A}\pi_1\pi_2\pi_2''$.

– $[\mathbf{LET}^h_{pm}]$ Equal to the $[\mathrm{LET}_m]$ case.

– $[\mathbf{LET}_p]$ The proof of this case follows the same ideas as the cases $[\mathbf{LET}_m]$ and $[\mathbf{LET}^X_{pm}]$.

Theorem 6 (Maximality of $\Vdash^\bullet$).

a) $\Pi^\bullet_{\mathcal{A},e}$ has a maximum element $\iff \exists\tau_g \in SType, \pi_g \in \mathcal{TS}ubst.\mathcal{A}\Vdash^\bullet e : \tau_g|\pi_g$.

b) If $\mathcal{A}\pi'\vdash^\bullet e:\tau'$ and $\mathcal{A}\Vdash^\bullet e:\tau|\pi$ then exists a type substitution π'' such that $\mathcal{A}\pi' = \mathcal{A}\pi\pi''$ and $\tau' = \tau\pi''$.

Proof.

a)

– $\Longleftarrow$) If $\mathcal{A}\Vdash^\bullet e:\tau_g|\pi_g$ then by Lemma 9 $\Pi_{\mathcal{A},e} = \Pi^\bullet_{\mathcal{A},e}$. Since $\mathcal{A}\Vdash e:\tau_g|\pi_g$ (by definition of $\Vdash^\bullet$) by Theorem 9 we know that $\Pi_{\mathcal{A},e}$ has a maximum element, and also $\Pi^\bullet_{\mathcal{A},e}$.

– $\Longrightarrow$) We will prove that $\mathcal{A} \not\Vdash^{\bullet} e : \tau_g|\pi_g \Longrightarrow \Pi^{\bullet}_{\mathcal{A},e}$ has not a maximum element.

(A) $\mathcal{A} \not\Vdash^{\bullet} e : \tau_g|\pi_g$ because $\mathcal{A} \not\Vdash e : \tau_g|\pi_g$. We know from Theorem 9 that if $\mathcal{A} \not\Vdash e : \tau_g|\pi_g$ then $\Pi_{\mathcal{A},e}$ has not a maximum element. Then by Theorem 5 it cannot exists any type derivation $\mathcal{A}\pi' \vdash e : \tau'$, so $\Pi_{\mathcal{A},e}$ is empty. Since $\Pi^{\bullet}_{\mathcal{A},e} \subseteq \Pi_{\mathcal{A},e}$ then $\Pi^{\bullet}_{\mathcal{A},e} = \emptyset$ and cannot contain any maximum element.

(B) $\mathcal{A} \not\Vdash^{\bullet} e : \tau_g|\pi_g$ because $\mathcal{A} \Vdash e : \tau_g|\pi_g$ but $critVar_{\mathcal{A}\pi_g}(e) \neq \emptyset$. We will proceed by case distintion over the cause of the critical variables:

(B.1) $critVar_{\mathcal{A}\pi_g}(e) \neq \emptyset$ because for every pattern t_j in e and for every variable X_i in t_j that is critical then the cause of the opacity are type variables which appear in $\mathcal{A}\pi_g$. In other words, for those variables X_i then $\mathcal{A} \oplus \{X_i : \alpha_i\} \Vdash t_j : \tau_j|\pi_j$ and $FTV(\alpha_i\pi_j) \not\subseteq FTV(\tau_j)$ and $FTV(\alpha_i\tau_j) \setminus FTV(\tau_j) \subseteq FTV(\mathcal{A}\pi_g)$. It is clear that we can apply a type substitution to $\mathcal{A}\pi_g$ and eliminate the opacity of these variables. In particular we will always be able to find two type substitions π_1 and π_2 such that:

i. $\mathcal{A}\pi_g\pi_1 \vdash e : \tau_1$ and $\mathcal{A}\pi_g\pi_2 \vdash e : \tau_2$.
ii. $critVar_{\mathcal{A}\pi_g\pi_1}(e) = \emptyset$ and $critVar_{\mathcal{A}\pi_g\pi_2}(e) = \emptyset$
iii. No substitution π more general than $\pi_g\pi_1$ and $\pi_g\pi_2$ is in $\Pi^{\bullet}_{\mathcal{A},e}$ because $critVar_{\mathcal{A}\pi}(e) = \emptyset$.

Let be $\overline{\beta_k}$ all the type variables causing opacity, and τ^1 and τ^2 two non unifiable types (*bool* and *char*, for example). Then we can define $\pi_1 \equiv [\overline{\beta_k/\tau^1}]$ and $\pi_2 \equiv [\overline{\beta_k/\tau^2}]$. Since $\mathcal{A} \Vdash e : \tau_g|\pi_g$ by Theorem 4 $\mathcal{A}\pi_g \vdash e : \tau_g$, and by Theorem 1-a $\mathcal{A}\pi_g\pi_1 \vdash e : \tau_g\pi_1$ and $\mathcal{A}\pi_g\pi_2 \vdash e : \tau_g\pi_2$. We have eliminated the cause of opacity, so $critVar_{\mathcal{A}\pi_g\pi_1}(e) = \emptyset$ and $critVar_{\mathcal{A}\pi_g\pi_2}(e) = \emptyset$, i.e., $\pi_g\pi_1, \pi_g\pi_2 \in \Pi^{\bullet}_{\mathcal{A},e}$. Finally since τ^1 and τ^2 are not unifiable, the only substitution more general that $\pi_g\pi_1$ and $\pi_g\pi_2$ that could be in $\Pi^{\bullet}_{\mathcal{A},e}$ is π_g (substitutions more general than π_g cannot be in $\Pi_{\mathcal{A},e}$, and neither in $\Pi^{\bullet}_{\mathcal{A},e}$). But π_g is not in $\Pi^{\bullet}_{\mathcal{A},e}$ because $critVar_{\mathcal{A}\pi_g}(e) \neq \emptyset$. Therefore $\Pi^{\bullet}_{\mathcal{A},e}$ cannot have a maximum element because we have found two elements in $\Pi^{\bullet}_{\mathcal{A},e}$ that do not have any "greater" element in $\Pi^{\bullet}_{\mathcal{A},e}$.

(B.2) $critVar_{\mathcal{A}\pi_g}(e) \neq \emptyset$ because there exists some pattern t_j in e in which there is any variable X that is opaque because of type variables that do not occur in $\mathcal{A}\pi_g$. Intuitively in this case these type variables will have appeared because of there exist a symbol in t_j whose type is a type-scheme, and that fresh variables come from the fresh variant used. From Theorem 5 we know that for every π_e in $\Pi_{\mathcal{A},e}$ then $\mathcal{A}\pi_e = \mathcal{A}\pi_g\pi''$ for some type substitution π'. But $critVar_{\mathcal{A}\pi_e}(e) = critVar_{\mathcal{A}\pi_g\pi''}(e) \neq \emptyset$, because we always have fresh type variables causing opacity (since they come from type-schemes, substitutions do not affect them). Therefore for every $\pi_e \in \Pi_{\mathcal{A},e}$ then $critVar_{\mathcal{A}\pi_e}(e) \neq \emptyset$, and as $\Pi^{\bullet}_{\mathcal{A},e} \subseteq \Pi_{\mathcal{A},e}$ then $\Pi^{\bullet}_{\mathcal{A},e} = \emptyset$; so it has not a maximum element.

---

b) By definition of $\vdash^\bullet$ and $\Vdash^\bullet$ we know that $\mathcal{A}\pi' \vdash e : \tau'$ and $\mathcal{A} \Vdash e : \tau|\pi$. Then by Theorem 5 we know that exists a type substitution π'' such that $\mathcal{A}\pi' = \mathcal{A}\pi\pi''$ and $\tau' = \tau\pi''$.

Theorem 7 (Soundness of $\mathcal{B}$).
$\mathcal{B}(\mathcal{A}, \mathcal{P}) = \pi \Longrightarrow wt_{\mathcal{A}\pi}(\mathcal{P})$.

Proof. From $\mathcal{B}(\mathcal{A}, \mathcal{P}) = \pi$ we have $\mathcal{A} \Vdash^\bullet (\varphi(r_1), \ldots, \varphi(r_m)) : (\tau_1, \ldots, \tau_m)|\pi$, and by Theorem 4 then $\mathcal{A}\pi \vdash^\bullet (\varphi(r_1), \ldots, \varphi(r_m)) : (\tau_1, \ldots, \tau_m)$. In order to prove $wt_{\mathcal{A}\pi}(\mathcal{P})$ we need to prove that every rule $r_i \equiv f_i\ t_1 \ldots t_n \rightarrow e_i$ in $\mathcal{P}$ is well-typed wrt. $\mathcal{A}\pi$. From Lemma 10 we know that $\mathcal{A}\pi \vdash^\bullet \varphi(r_i) : \tau_i$, so $\mathcal{A}\pi \vdash^\bullet pair\ \lambda t_1 \ldots t_n.e_i\ f_i : \tau_i$. This derivation can only be constructed if $\mathcal{A}\pi \vdash^\bullet \lambda t_1 \ldots t_n.e_i : \tau_i$ and $\mathcal{A}\pi \vdash^\bullet f_i : \tau_i$, and as the last derivation is just an application of rule **[ID]**, $\mathcal{A}\pi(f_i) \succ \tau_i$. We will distinguish between the case that $\mathcal{A}(f_i)$ is a simple type or a closed type-scheme:

a) If $\mathcal{A}(f_i)$ is a simple type, then $\mathcal{A}\pi(f_i)$ too. In this case $\mathcal{A}\pi(f_i) \succ \tau_i$ can only be true if $\mathcal{A}\pi(f_i) = \tau_i$, so trivially τ_i is a variant of $\mathcal{A}\pi(f_i)$. Therefore $\mathcal{A}\pi \vdash^\bullet \lambda t_1 \ldots t_n.e_i : \tau_i$ and τ_i is a variant of $\mathcal{A}\pi(f_i)$, so rule r_i is well-typed wrt. $\mathcal{A}\pi$.
b) $\mathcal{A}(f_i)$ is a closed type scheme, so $\mathcal{A}(f_i) = \mathcal{A}\pi(f_i)$. From step 2.- of $\mathcal{B}$ we know that in this case τ_i is a variant of $\mathcal{A}(f_i)$, and also of $\mathcal{A}\pi(f_i)$. Then since $\mathcal{A}\pi \vdash^\bullet \lambda t_1 \ldots t_n.e_i : \tau_i$ rule r_i is well-typed wrt. $\mathcal{A}\pi$.

Theorem 8 (Maximality of $\mathcal{B}$).
If $wt_{\mathcal{A}\pi'}(\mathcal{P})$ *and* $\mathcal{B}(\mathcal{A}, \mathcal{P}) = \pi$ *then* $\exists\pi''$ *such that* $\mathcal{A}\pi' = \mathcal{A}\pi\pi''$.

Proof. Since $wt_{\mathcal{A}\pi'}(\mathcal{P})$ we know that for every rule $r_i \equiv f_i\ t_1 \ldots t_n \rightarrow e_i$ in $\mathcal{P}$ there exists a type derivation $\mathcal{A}\pi' \vdash^\bullet \lambda t_1 \ldots t_n.e_i : \tau_i'$ and τ_i' is a variant of the type $\mathcal{A}\pi'(f_i)$. Then $\mathcal{A}\pi' \vdash^\bullet f_i : \tau_i'$, and we can construct type derivations $\mathcal{A}\pi' \vdash^\bullet pair\ \lambda t_1 \ldots t_n.e_i\ f_i : \tau_i'$. With these derivations we can build $\mathcal{A}\pi' \vdash^\bullet (\varphi(r_1), \ldots, \varphi(r_m)) : (\tau_1', \ldots, \tau_m')$ by Lemma 10. From $\mathcal{B}(\mathcal{A}, \mathcal{P}) = \pi$ we know that $\mathcal{A} \Vdash^\bullet (\varphi(r_1), \ldots, \varphi(r_m)) : (\tau_1, \ldots, \tau_m)|\pi$, so by Theorem 6-*b* there will exist some type substitution π'' such that $\mathcal{A}\pi' = \mathcal{A}\pi\pi''$.

Theorem 9 (Maximality of $\Vdash$).
$\Pi_{\mathcal{A},e}$ *has a maximum element* $\pi \Longleftrightarrow \exists\tau_g, \pi_g \in SType.\mathcal{A} \Vdash e : \tau_g|\pi_g$.

Proof.

$\Longrightarrow$) If $\Pi_{\mathcal{A},e}$ has maximum element π then there will be some type τ such that $\mathcal{A}\pi \vdash e : \tau$. Then by Theorem 5 we know that $\mathcal{A} \Vdash e : \tau_g|\pi_g$.
$\Longleftarrow$) We know from Theorem 5 that for every type substitution $\pi' \in \Pi_{\mathcal{A},e}$ there exists a type substitution π'' such that $\mathcal{A}\pi' = \mathcal{A}\pi\pi''$. Then $\pi|_{FTV(\mathcal{A})} \lesssim \pi'$. From Theorem 4 we know that $\pi|_{FTV(\mathcal{A})}$ is in $\Pi_{\mathcal{A},e}$, so it is the maximum element.