A EXTENSIONS

In this appendix, we present some extensions for the source language, in particular let-bindings (not to be coufounded with the top-level definitions composing a program: the let-bindings presented in this section can be used anywhere in an expression and do not generalize the type of their definition) and pattern matching.

This section gives an overview of these extensions together with some explanations, but the full semantics and typing rules can be found in the next appendices.

A.1 Let Bindings

A.1.1 Declarative Type System. Let bindings can be added to the syntax of our language:

$$\textbf{Expressions} \quad e \quad ::= \quad \cdots \mid \texttt{let}\, x = e \,\texttt{in}\, e \tag{4}$$

with the following notion of reduction:

$$\texttt{let}\, x = v \,\texttt{in}\, e \quad \leadsto \quad e\{v/x\}$$

At first sight, we could think of adding this typing rule to the declarative type system:

$$[\textsc{Let}]\ \frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma, (x : t_1) \vdash e_2 : t_2}{\Gamma \vdash \texttt{let}\, x = e_1 \,\texttt{in}\, e_2 \ : t_2}$$

However, this extension of the declarative type system has one issue: let-bindings can introduce aliasing, preventing in some cases the [$\vee$] rule from applying. For instance, consider the following expression:

$$\lambda x.\ \texttt{let}\, y = x \,\texttt{in}\, (f\ x \in \mathsf{Int})\ ?\ f\ y : 42$$

with $f : \mathbb{1} \to \mathbb{1}$.

Though for any argument x this function yields an integer, it is not possible to derive for it the type $\mathbb{1} \to \mathsf{Int}$ using this extension of the declarative type system. Indeed, $f\ x$ and $f\ y$ are not syntactically equivalent and thus the [$\vee$] rule can only decompose their types independently, loosing the correlation between these two expressions.

One way to fix this issue is to remove this kind of aliasing before applying the declarative type system. For that, we can introduce an intermediate language featuring an alternative version of let-bindings:

$$\textbf{Expressions} \quad e \quad ::= \quad \cdots \mid \texttt{let}\, e \,\texttt{in}\, e \tag{5}$$

Let-bindings of the source language can be transformed into this alternative version using a transfomration $[\![.]\!]$ defined as follows (the other cases are straightforward):

$$[\![\texttt{let}\, x = e_1 \,\texttt{in}\, e_2\,]\!] = \texttt{let}\ [\![e_1]\!] \ \texttt{in}\ [\![e_2]\!]\{[\![e_1]\!]/x\}$$

Finally, the declarative type system can be extended with this rule:

$$[\textsc{Let}]\ \frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma \vdash e_2 : t_2}{\Gamma \vdash \texttt{let}\, e_1 \,\texttt{in}\, e_2 \ : t_2}$$

A.1.2 Algorithmic type system. Let-bindings are added to MSC forms as a new atom construction:

$$\textbf{Atomic expr} \quad a \quad ::= \quad \cdots \mid \mathtt{let}\,\mathtt{x}\,\mathtt{in}\,\mathtt{x} \tag{6}$$

The intuition is the same as for the declarative type system: we want to get rid of the aliasing caused by let-bindings, while still using bindings to *factorize* each subexpression. Indeed, to produce an atom for the expression $\mathtt{let}\,x = e_1\,\mathtt{in}\,e_2$ we must replace each subexpression by a binding variable, which would yield something of the form $\mathtt{let}\,x = \mathtt{x}_1\,\mathtt{in}\,\mathtt{x}_2$. Since the body of the let-expression is a variable, then the variable x is only an alias for $\mathtt{x}_1$ and thus is undesirable. Consequently, only the other two variables are specified, which yields $\mathtt{let}\,\mathtt{x}_1\,\mathtt{in}\,\mathtt{x}_2$ and which explains the definition of the atom for let expressions.

For instance, the example expression $\mathtt{let}\,x = \lambda y.y\,\mathtt{in}\,(x, x)$ has the following canonical form:

$$\begin{array}{l} \mathtt{bind}\,\mathtt{x}_1 = \lambda y.\mathtt{bind}\,\mathtt{y} = y\,\mathtt{in}\,\mathtt{y}\ \mathtt{in} \\ \mathtt{bind}\,\mathtt{x}_2 = (\mathtt{x}_1, \mathtt{x}_1)\ \mathtt{in} \\ \mathtt{bind}\,\mathtt{x}_\circ = (\mathtt{let}\,\mathtt{x}_1\,\mathtt{in}\,\mathtt{x}_2\,)\ \mathtt{in}\,\mathtt{x}_\circ \end{array}$$

Note that, as explained above, the variable x is no longer present in the canonical form.

The algorithmic type system can then be extended with the following rule:

$$[\textsc{Let-Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [\mathtt{let}\,\mathtt{x}_1\,\mathtt{in}\,\mathtt{x}_2 \mid \varnothing] : \Gamma(\mathtt{x}_2)}\ \mathtt{x}_1 \in \mathsf{dom}(\Gamma)$$

It is straightforward to extend the reconstruction with additional rules in order to support this new construction (c.f. appendix H).

A.2 Type constraints

A new construction $(e \mathbin{⨾} \tau)$ can be added to our source language. This construction acts as a type constraint: if the expression e does not reduce to a value of type τ (and does not diverge), then the reduction will be stuck. In a sense, it could be seen as a *cast*, but we will not use this terminology in order to avoid confusions with gradual typing. Actually, we only introduce this construction because it will be used later to encode more general type-cases.

We add the following construction to our source language:

$$\textbf{Expressions} \quad e \quad ::= \quad \cdots \mid (e \mathbin{⨾} \tau) \tag{7}$$

with the following notion of reduction:

$$(v \mathbin{⨾} \tau) \quad \rightsquigarrow \quad v \qquad\qquad \text{if } v \in \tau$$

The declarative type system can trivially be extended by adding this rule:

$$[\textsc{Constr}]\ \frac{\Gamma \vdash e \mathbin{⨾} \tau \qquad \Gamma \vdash e : t}{\Gamma \vdash (e \mathbin{⨾} \tau) : t}$$

The same construction is added to the atoms of canonical forms:

$$\textbf{Atomic expr} \quad a \quad ::= \quad \cdots \mid \mathtt{x} \mathbin{⨾} \tau \tag{8}$$

The annotations of the algorithmic type system also need to be extended:

$$\textbf{Atoms annotations} \quad \mathbb{a} \quad ::= \quad \cdots \mid ⨾(\Sigma) \tag{9}$$

and the algorithmic type system is extended with the following rule:

$$[\textsc{Constr-Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [\mathtt{x} \mathbin{⨾} \tau \mid ⨾(\Sigma)] : \Gamma(\mathtt{x})}\ \Gamma(\mathtt{x})\Sigma \leq \tau$$

It is also straightforward to extend the reconstruction with additional rules in order to support this new construction (c.f. appendix H).

A.3 Pattern matching

Pattern matching is a fundamental feature of functional languages, and even some dynamic languages such as Python have started to implement it. In this section, we show how this feature can be added in our source language. We proceed in two steps: first, a more general typecase construct with arbitrary arity is introduced, and secondly, this construct is generalized again so that branches can be decorated with patterns instead of just types.

A.3.1 Extended typecases. We start by adding a generalized version of the typecase, that can have any number of branches:

$$\textbf{Expressions} \quad e \quad ::= \quad \cdots \mid (\texttt{tcase}\, e \,\texttt{of}\, \tau \to e \mid \ldots \mid \tau \to e) \tag{10}$$

with the following notion of reduction:

$$\texttt{tcase}\, v \,\texttt{of}\, \tau_1 \to e_1 \mid \ldots \mid \tau_n \to e_n \quad \rightsquigarrow \quad e_k \quad \begin{array}{l}\text{if } v : \tau_k \setminus (\bigvee_{i\in 1..k-1} \tau_i)\\ \text{for any } k \in 1\,..\,n\end{array} \tag{11}$$

In term of typing, however, we choose not to extend the type system with additional rules in order to preserve its minimality. Instead, we transform expressions with extended typecases into expressions of the source langauge presented in section 2, with the let-binding and type constraints extensions (A.1 and A.2). For that, we use the following transformation:

$$\begin{aligned}
(\!| c |\!) &= c\\
(\!| x |\!) &= x\\
(\!| \lambda x.e |\!) &= \lambda x.(\!| e |\!)\\
(\!| \pi_i e |\!) &= \pi_i (\!| e |\!)\\
(\!| e_1 e_2 |\!) &= (\!| e_1 |\!)(\!| e_2 |\!)\\
(\!| (e_1, e_2) |\!) &= ((\!| e_1 |\!), (\!| e_2 |\!))\\
(\!| (e{\in}\tau)\, ?\, e_1 : e_2 |\!) &= ((\!| e |\!){\in}\tau)\, ?\, (\!| e_1 |\!) : (\!| e_2 |\!)\\
(\!| \texttt{let}\, x = e_1 \,\texttt{in}\, e_2 |\!) &= \texttt{let}\, x = (\!| e_1 |\!) \,\texttt{in}\, (\!| e_2 |\!)\\
(\!| e \fatsemi \tau |\!) &= (\!| e |\!) \fatsemi \tau\\
(\!| \texttt{tcase}\, e \,\texttt{of}\, \tau_1 \to e_1 \mid \ldots \mid \tau_n \to e_n |\!) &= \begin{array}{l}\texttt{let}\, x = ((\!| e |\!) \fatsemi \bigvee_{i\in 1..n} \tau_i) \,\texttt{in}\\ \mathsf{c}_x(\tau_1 \to (\!| e_1 |\!)\,;\, \ldots\,;\, \tau_n \to (\!| e_n |\!))\end{array} \quad \text{with } x \text{ fresh}\\
\mathsf{c}_x(\tau \to e) &= e\\
\mathsf{c}_x(\tau \to e\,;\, C) &= (x{\in}\tau)\, ?\, e : \mathsf{c}_x(C)
\end{aligned}$$

A.3.2 Pattern matching. Now, we introduce patterns and a pattern matching construct in the source language:

$$\begin{array}{llll}\textbf{Patterns} & p & ::= & \tau \mid x \mid p\&p \mid p|p \mid (p,p) \mid x := c\\ \textbf{Expressions} & e & ::= & \cdots \mid (\texttt{match}\, e \,\texttt{with}\, p \to e \mid \ldots \mid p \to e)\end{array} \tag{12}$$

The associated reduction rule can be found in Appendix B.

In term of typing, we proceed as before by transforming an expression with pattern matching into an expression without pattern matching (but with extended typecases and let-bindings), using the following transformation:

$$
\begin{aligned}
(\!| c |\!) &= c\\
(\!| x |\!) &= x\\
(\!| \lambda x.e |\!) &= \lambda x.(\!| e |\!)\\
(\!| \pi_i e |\!) &= \pi_i (\!| e |\!)\\
(\!| e_1 e_2 |\!) &= (\!| e_1 |\!)(\!| e_2 |\!)\\
(\!| (e_1, e_2) |\!) &= ((\!| e_1 |\!), (\!| e_2 |\!))\\
(\!| (e {\in} \tau)\ ?\ e_1 : e_2 |\!) &= ((\!| e |\!) {\in} \tau)\ ?\ (\!| e_1 |\!) : (\!| e_2 |\!)\\
(\!| \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 |\!) &= \mathtt{let}\ x = (\!| e_1 |\!)\ \mathtt{in}\ (\!| e_2 |\!)\\
(\!| e \mathbin{⨟} \tau |\!) &= (\!| e |\!) \mathbin{⨟} \tau\\
(\!| \mathtt{tcase}\ e\ \mathtt{of}\ \tau_1 \to e_1 \mid \dots \mid \tau_n \to e_n |\!) &= \mathtt{tcase}\ (\!| e |\!)\ \mathtt{of}\ \tau_1 \to (\!| e_1 |\!) \mid \dots \mid \tau_n \to (\!| e_n |\!)\\
(\!| \mathtt{match}\ e\ \mathtt{with}\ p_1 \to e_1 \mid \dots \mid p_n \to e_n |\!) &= \mathtt{let}\ x = (\!| e |\!)\ \mathtt{in}\ \mathtt{tcase}\ x\ \mathtt{of}\ \begin{array}{l} \wr p_1 \int \to e'_1\\ \mid \dots\\ \mid \wr p_n \int \to e'_n \end{array}
\end{aligned}
$$

with x fresh, and where for every $i \in 1 \mathinner{..} m$:
$e'_i = \mathtt{let}\ x_1 = \mathrm{d}_{x_1}(p_i, x)\ \mathtt{in}\ \dots \mathtt{let}\ x_m = \mathrm{d}_{x_m}(p_i, x)\ \mathtt{in}\ (\!| e_i |\!)$ for $\{x_1, ..., x_m\} = \mathsf{vars}(p_i)$ with

$$
\begin{aligned}
\mathrm{d}_x(x, e) &= e\\
\mathrm{d}_x(x := c, e) &= c\\
\mathrm{d}_x((p_1, p_2), e) &= \mathrm{d}_x(p_i, \pi_i e) && \text{if } x \in \mathsf{vars}(p_i)\\
\mathrm{d}_x(p_1 \& p_2, e) &= \mathrm{d}_x(p_i, e) && \text{if } x \in \mathsf{vars}(p_i)\\
\mathrm{d}_x(p_1 | p_2, e) &= (e {\in} \wr p_1 \int)\ ?\ \mathrm{d}_x(p_1, e) : \mathrm{d}_x(p_2, e)\\
\mathrm{d}_x(p, e) &= \text{undefined} && \text{otherwise}
\end{aligned}
$$

B FULL SEMANTICS WITH EXTENSIONS

Expressions of the source language with extensions of Appendix A are defined as follows:

$$
\begin{array}{llll}
\textbf{Test Types} & \tau & ::= & b \mid \mathbb{0} \to \mathbb{1} \mid \tau \times \tau \mid \tau \vee \tau \mid \neg\tau \mid \mathbb{0}\\
\textbf{Patterns} & p & ::= & \tau \mid x \mid p \& p \mid p | p \mid (p, p) \mid x := c\\
\textbf{Expressions} & e & ::= & c \mid x \mid \lambda x.e \mid ee \mid (e, e) \mid \pi_i e \mid (e {\in} \tau)\ ?\ e : e \mid \mathtt{let}\ x = e\ \mathtt{in}\ e \mid (e \mathbin{⨟} \tau)\\
 & & & \mid (\mathtt{tcase}\ e\ \mathtt{of}\ \tau \to e \mid \dots \mid \tau \to e) \mid (\mathtt{match}\ e\ \mathtt{with}\ p \to e \mid \dots \mid p \to e)\\
\textbf{Values} & v & ::= & c \mid \lambda x.e \mid (v, v)
\end{array}
$$

The associated reduction rules are:

$$
\begin{array}{rcll}
(\lambda x.e)v & \rightsquigarrow & e\{v/x\} & & (13)\\
\pi_1(v_1,v_2) & \rightsquigarrow & v_1 & & (14)\\
\pi_2(v_1,v_2) & \rightsquigarrow & v_2 & & (15)\\
(v{\in}\tau)\,?\,e_1 : e_2 & \rightsquigarrow & e_1 & \text{if } v \in \tau & (16)\\
(v{\in}\tau)\,?\,e_1 : e_2 & \rightsquigarrow & e_2 & \text{if } v \in \neg\tau & (17)\\
\texttt{let}\, x = v\, \texttt{in}\, e & \rightsquigarrow & e\{v/x\} & & (18)\\
(v \mathbin{⨾} \tau) & \rightsquigarrow & v & \text{if } v \in \tau & (19)\\
\texttt{tcase}\, v\, \texttt{of}\, \tau_1 \to e_1 \mid \ldots \mid \tau_n \to e_n & \rightsquigarrow & e_k & \begin{array}{l}\text{if } v : \tau_k \setminus (\bigvee_{i\in 1..k-1} \tau_i)\\ \text{for any } k \in 1..n\end{array} & (20)\\
\texttt{match}\, v\, \texttt{with}\, p_1 \to e_1 \mid \ldots \mid p_n \to e_n & \rightsquigarrow & e_k(v/p_k) & \begin{array}{l}\text{if } v : \{\!\!\{ p_k \}\!\!\} \setminus (\bigvee_{i\in 1..k-1} \{\!\!\{ p_i \}\!\!\})\\ \text{for any } k \in 1..n\end{array} & (21)
\end{array}
$$

together with the context rules that implement a leftmost outermost reduction strategy, that is, $E[e] \rightsquigarrow E[e']$ if $e \rightsquigarrow e'$ where the evaluation contexts $E[\,]$ are defined as follows:

$$
\begin{array}{lrl}
\textbf{Evaluation Context} & E ::= & [\,] \mid vE \mid Ee \mid (v,E) \mid (E,e) \mid \pi_i E \mid (E{\in}\tau)\,?\,e : e\\
& & \mid \texttt{let}\, x = E\, \texttt{in}\, e \mid (E \mathbin{⨾} \tau) \mid (\texttt{tcase}\, E\, \texttt{of}\, \ldots) \mid (\texttt{match}\, E\, \texttt{with}\, \ldots)
\end{array}
$$

Capture-avoiding substitutions are defined as follows (cases for extended typecases and pattern-matchings have been omitted for concision):

$$
\begin{array}{rcll}
c\{e'/x\} & = & c & & (22)\\
x\{e'/x\} & = & e' & & (23)\\
y\{e'/x\} & = & y & x \neq y & (24)\\
(\lambda x.e)\{e'/x\} & = & \lambda x.e & & (25)\\
(\lambda y.e)\{e'/x\} & = & \lambda y.(e\{e'/x\}) & x \neq y, y \notin \mathrm{fv}(e') & (26)\\
(\lambda y.e)\{e'/x\} & = & \lambda z.(e\{z/y\}\{e'/x\}) & x \neq y, y \in \mathrm{fv}(e'), z \text{ fresh} & (27)\\
(e_1e_2)\{e'/x\} & = & (e_1\{e'/x\})(e_2\{e'/x\}) & & (28)\\
(e_1,e_2)\{e'/x\} & = & (e_1\{e'/x\}, e_2\{e'/x\}) & & (29)\\
(\pi_i e)\{e'/x\} & = & \pi_i(e\{e'/x\}) & & (30)\\
((e_1{\in}\tau)\,?\,e_2 : e_3)\{e'/x\} & = & (e_1\{e'/x\}{\in}\tau)\,?\,e_2\{e'/x\} : e_3\{e'/x\} & & (31)\\
(\texttt{let}\, x = e_1\, \texttt{in}\, e_2)\{e'/x\} & = & \texttt{let}\, x = e_1\{e'/x\}\, \texttt{in}\, e_2 & & (32)\\
(\texttt{let}\, y = e_1\, \texttt{in}\, e_2)\{e'/x\} & = & \texttt{let}\, y = e_1\{e'/x\}\, \texttt{in}\, e_2\{e'/x\} & x \neq y, y \notin \mathrm{fv}(e') & (33)\\
(\texttt{let}\, y = e_1\, \texttt{in}\, e_2)\{e'/x\} & = & \texttt{let}\, y = e_1\{e'/x\}\, \texttt{in}\, e_2\{z/y\}\{e'/x\} & x \neq y, y \in \mathrm{fv}(e'), z \text{ fresh} & (34)
\end{array}
$$

The relation $v \in \tau$ that determines whether a value is of a given type or not and holds true if and only if $\mathrm{typeof}(v) \leq \tau$, where

$$
\begin{array}{rcl}
\mathrm{typeof}(\lambda x.e) & = & \mathbb{0} \to \mathbb{1}\\
\mathrm{typeof}(c) & = & b_c\\
\mathrm{typeof}((v_1,v_2)) & = & \mathrm{typeof}(v_1) \times \mathrm{typeof}(v_2)
\end{array}
$$

Finally, the operators used in the reduction rule for pattern matching are defined as follows:

$$\begin{aligned}
\wr\tau\int &= \tau\\
\wr x\int &= \mathbb{1}\\
\wr p_1\&p_2\int &= \wr p_1\int \wedge \wr p_2\int\\
\wr p_1|p_2\int &= \wr p_1\int \vee \wr p_2\int\\
\wr(p_1,p_2)\int &= \wr p_1\int \times \wr p_2\int\\
\wr x := c\int &= \mathbb{1}
\end{aligned}$$

and

$$\begin{array}{rlr}
v/\tau = & \mathtt{id} & \text{if } v:\tau\\
v/x = & \{v/x\} & \\
v/(p_1\&p_2) = & \sigma_1\cup\sigma_2 & \text{if } \sigma_1 = v/p_1 \text{ and } \sigma_2 = v/p_2\\
v/(p_1|p_2) = & v/p_1 & \text{if } v/p_1 \neq \mathtt{fail}\\
v/(p_1|p_2) = & v/p_2 & \text{if } v/p_1 = \mathtt{fail}\\
v/(p_1,p_2) = & \sigma_1\cup\sigma_2 & \text{if } v=(v_1,v_2),\ \ \sigma_1 = v_1/p_1 \text{ and } \sigma_2 = v_2/p_2\\
v/(x := c) = & \{c/x\} & \\
v/p = & \mathtt{fail} & \text{otherwise}
\end{array}$$

C SUBTYPING RELATION

Subtyping is defined by giving a set-theoretic interpretation of the types of Definition 2.1 into a suitable domain $\mathcal{D}$. In case of polymorphic types, the domain at issue must satisfy the property of *convexity* [Castagna and Xu 2011]. A simple model that satisfies convexity was proposed by [Gesbert et al. 2015]. We succintly present it in this section. The reader my refer to [Castagna 2023a, Section 3.3] for more details.

Definition C.1 (Interpretation domain [Gesbert et al. 2015]). *The* interpretation domain *$\mathcal{D}$ is the set of finite terms d produced inductively by the following grammar*

$$\begin{aligned}
d &::= c^L \mid (d,d)^L \mid \{(d,\partial),\dots,(d,\partial)\}^L\\
\partial &::= d \mid \Omega
\end{aligned}$$

where c ranges over the set $\mathcal{C}$ of constants, L ranges over finite sets of type variables, and where Ω is such that $\Omega\notin\mathcal{D}$.

The elements of $\mathcal{D}$ correspond, intuitively, to (denotations of) the results of the evaluation of expressions, labeled by finite sets of type variables. In particular, in a higher-order language, the results of computations can be functions which, in this model, are represented by sets of finite relations of the form $\{(d_1,\partial_1),\dots,(d_n,\partial_n)\}^L$, where Ω (which is not in $\mathcal{D}$) can appear in second components to signify that the function fails (i.e., evaluation is stuck) on the corresponding input. This is implemented by using in the second projection the meta-variable ∂ which ranges over $\mathcal{D}_\Omega = \mathcal{D}\cup\{\Omega\}$ (we reserve d to range over $\mathcal{D}$, thus excluding Ω). This constant Ω is used to ensure that $\mathbb{1}\to\mathbb{1}$ is not a supertype of all function types: if we used d instead of ∂, then every well-typed function could be subsumed to $\mathbb{1}\to\mathbb{1}$ and, therefore, every application could be given the type $\mathbb{1}$, independently from its argument as long as this argument is typable (see Section 4.2 of [Frisch et al. 2008] for details). The restriction to *finite* relations corresponds to the intuition that the denotational semantics of a function is given by the set of its finite approximations, where finiteness is a restriction necessary (for cardinality reasons) to give the semantics to higher-order functions. Finally, the sets of type variables that label the elements of the domain are used to interpret type variables: we interpret a type variable α by the set of all elements that are labeled by α, that is $[\![\alpha]\!] = \{d \mid \alpha \in \mathrm{tags}(d)\}$ (where we define $\mathrm{tags}(c^L) = \mathrm{tags}((d, d')^L) = \mathrm{tags}(\{(d_1, \partial_1), \ldots, (d_n, \partial_n)\}^L) = L$).

We define the interpretation $[\![t]\!]$ of a type t so that it satisfies the following equalities, where $\mathcal{P}_{\mathrm{fin}}$ denotes the restriction of the powerset to finite subsets and $\mathbb{B}$ denotes the function that assigns to each basic type the set of constants of that type, so that for every constant c we have $c \in \mathbb{B}(b_c)$ (we use b_c to denote the basic type of the constant c):

$$
\begin{array}{lll}
[\![\mathbb{0}]\!] = \varnothing & [\![\alpha]\!] = \{d \mid \alpha \in \mathrm{tags}(d)\} & [\![t_1 \vee t_2]\!] = [\![t_1]\!] \cup [\![t_2]\!] \\
[\![b]\!] = \mathbb{B}(b) & [\![\neg t]\!] = \mathcal{D} \setminus [\![t]\!] & [\![t_1 \times t_2]\!] = [\![t_1]\!] \times [\![t_2]\!]
\end{array}
$$

$$[\![t_1 \to t_2]\!] = \{R \in \mathcal{P}_{\mathrm{fin}}(\mathcal{D} \times \mathcal{D}_\Omega) \mid \forall (d, \partial) \in R.\ d \in [\![t_1]\!] \implies \partial \in [\![t_2]\!]\}$$

We cannot take the equations above directly as an inductive definition of $[\![]\!]$ because types are not defined inductively but coinductively. Notice however that the contractivity condition of Definition 2.1 ensures that the binary relation $\rhd \subseteq \mathbf{Types} \times \mathbf{Types}$ defined by $t_1 \vee t_2 \rhd t_i$, $t_1 \wedge t_2 \rhd t_i$, $\neg t \rhd t$ is Noetherian. This gives an induction principle[7] on **Types** that we use combined with structural induction on $\mathcal{D}$ to give the following definition, which validates these equalities.

Definition C.2 (Set-theoretic interpretation of types). *We define a binary predicate $(d : t)$ ("the element d belongs to the type t"), where $d \in \mathcal{D}$ and $t \in$ **Types**, by induction on the pair (d, t) ordered lexicographically. The predicate is defined as follows:*

$$
\begin{array}{rcll}
(c : b) & = & c \in \mathbb{B}(b) & \\
(d : \alpha) & = & \alpha \in tags(d) & \\
((d_1, d_2) : t_1 \times t_2) & = & (d_1 : t_1) \text{ and } (d_2 : t_2) & \\
(\{(d_1, \partial_1), ..., (d_n, \partial_n)\} : t_1 \to t_2) & = & \forall i \in [1..n].\ \text{if } (d_i : t_1) \text{ then } (\partial_i : t_2) & \\
(d : t_1 \vee t_2) & = & (d : t_1) \text{ or } (d : t_2) & \\
(d : \neg t) & = & \text{not } (d : t) & \\
(\partial : t) & = & \text{false} & \textit{otherwise}
\end{array}
$$

We define the set-theoretic interpretation $[\![]\!] : \mathbf{Types} \to \mathcal{P}(\mathcal{D})$ *as* $[\![t]\!] = \{d \in \mathcal{D} \mid (d : t)\}$.

Finally, we define the subtyping preorder and its associated equivalence relation as follows.

Definition C.3 (Subtyping relation). *We define the* subtyping *relation $\leq$ and the* subtyping equivalence *relation $\simeq$ as* $t_1 \leq t_2 \overset{\text{def}}{\iff} [\![t_1]\!] \subseteq [\![t_2]\!]$ *and* $t_1 \simeq t_2 \overset{\text{def}}{\iff} (t_1 \leq t_2) \text{ and } (t_2 \leq t_1)$.

D DECLARATIVE TYPE SYSTEM WITH EXTENSIONS

The declarative type system extended with the extensions of Appendix A uses expressions produced by the following grammar:

$$\textbf{Expressions} \quad e \quad ::= \quad c \mid x \mid \lambda x.e \mid ee \mid (e, e) \mid \pi_i e \mid (e {\in} \tau)\,?\,e : e \mid \mathtt{let}\ e\ \mathtt{in}\ e \mid (e \mathbin{⨾} \tau)$$

Note that extended typecases and pattern matching are absent because they are encoded using let-bindings and type constraints before typing. Similarly, we use the construction `let` e `in` e for let-bindings instead of the initial construction `let` $x = e$ `in` e in order to avoid aliasing. You should refer to Section A.1 for more details on this transformation.

[7]In a nutshell, we can do proofs and give definitions by induction on the structure of unions and negations—and, thus, intersections—but arrows, products, and basic types are the base cases for the induction.

The deduction rules for the declarative type system are:

$$[\textsc{Const}]\ \frac{}{\Gamma \vdash c : b_c} \qquad [\textsc{Ax}]\ \frac{}{\Gamma \vdash x : \Gamma(x)}$$

$$[\to\text{I}]\ \frac{\Gamma, x : \mathbf{u} \vdash e : t}{\Gamma \vdash \lambda x.e : \mathbf{u} \to t} \qquad [\to\text{E}]\ \frac{\Gamma \vdash e_1 : t_1 \to t_2 \qquad \Gamma \vdash e_2 : t_1}{\Gamma \vdash e_1 e_2 : t_2}$$

$$[\times\text{I}]\ \frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma \vdash e_2 : t_2}{\Gamma \vdash (e_1, e_2) : t_1 \times t_2} \qquad [\times\text{E}_1]\ \frac{\Gamma \vdash e : t_1 \times t_2}{\Gamma \vdash \pi_1 e : t_1} \qquad [\times\text{E}_2]\ \frac{\Gamma \vdash e : t_1 \times t_2}{\Gamma \vdash \pi_2 e : t_2}$$

$$[\mathbb{0}]\ \frac{\Gamma \vdash e : \mathbb{0}}{\Gamma \vdash (e\in\tau)\,?\,e_1 : e_2 : \mathbb{0}} \qquad [\in_1]\ \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash e_1 : t_1}{\Gamma \vdash (e\in\tau)\,?\,e_1 : e_2 : t_1} \qquad [\in_2]\ \frac{\Gamma \vdash e : \neg\tau \qquad \Gamma \vdash e_2 : t_2}{\Gamma \vdash (e\in\tau)\,?\,e_1 : e_2 : t_2}$$

$$[\vee]\ \frac{\Gamma \vdash e' : s \qquad \Gamma, x : s \wedge \mathbf{u} \vdash e : t \qquad \Gamma, x : s \wedge \neg\mathbf{u} \vdash e : t}{\Gamma \vdash e\{e'/x\} : t} \qquad [\wedge]\ \frac{\Gamma \vdash e : t_1 \qquad \Gamma \vdash e : t_2}{\Gamma \vdash e : t_1 \wedge t_2}$$

$$[\textsc{Inst}]\ \frac{\Gamma \vdash e : t}{\Gamma \vdash e : t\sigma} \qquad [\leq]\ \frac{\Gamma \vdash e : t}{\Gamma \vdash e : t'}\ t \leq t'$$

with these additional rules for the extensions of Appendix A (let-bindings and type constraints):

$$[\textsc{Let}]\ \frac{\Gamma \vdash e_1 : t_1 \qquad \Gamma \vdash e_2 : t_2}{\Gamma \vdash \mathtt{let}\, e_1\, \mathtt{in}\, e_2\ : t_2} \qquad [\textsc{Constr}]\ \frac{\Gamma \vdash e \mathbin{⨾} \tau \qquad \Gamma \vdash e : t}{\Gamma \vdash (e \mathbin{⨾} \tau) : t}$$

E COMPUTATION OF MSC-FORMS

E.1 From canonical forms to source language expressions

We recall the grammar for canonical forms, with the extensions presented in Appendix A:

$$\begin{array}{llcl}
\textbf{Atomic expressions} & a & ::= & c \mid x \mid \lambda x.\kappa \mid (\mathsf{x}, \mathsf{x}) \mid \mathsf{xx} \mid \pi_i\mathsf{x} \mid (\mathsf{x}\in\tau)\,?\,\mathsf{x} : \mathsf{x} \mid \mathtt{let}\,\mathsf{x}\,\mathtt{in}\,\mathsf{x} \mid \mathsf{x} \mathbin{⨾} \tau \\
\textbf{Canonical Forms} & \kappa & ::= & \mathsf{x} \mid \mathtt{bind}\,\mathsf{x} = a\ \mathtt{in}\,\kappa
\end{array}$$

Any canonical form can be transformed into an expression of the source language using the unwiding operator $\lceil . \rceil$ defined as follows:

$$\begin{array}{rcll}
\lceil c \rceil & = & c & \\
\lceil x \rceil & = & x & \\
\lceil \lambda x.\kappa \rceil & = & \lambda x.\lceil \kappa \rceil & \\
\lceil \mathsf{x}_1\mathsf{x}_2 \rceil & = & \mathsf{x}_1\mathsf{x}_2 & \\
\lceil (\mathsf{x}_1, \mathsf{x}_2) \rceil & = & (\mathsf{x}_1, \mathsf{x}_2) & \\
\lceil \pi_i\mathsf{x} \rceil & = & \pi_i\mathsf{x} & i = 1, 2 \\
\lceil (\mathsf{x}\in\tau)\,?\,\mathsf{x}_1 : \mathsf{x}_2 \rceil & = & (\mathsf{x}\in\tau)\,?\,\mathsf{x}_1 : \mathsf{x}_2 & \\
\lceil \mathtt{let}\,\mathsf{x}_1\,\mathtt{in}\,\mathsf{x}_2 \rceil & = & \mathtt{let}\, x = \mathsf{x}_1\,\mathtt{in}\,\mathsf{x}_2\{x/\mathsf{x}_1\}\ \text{ with } x \text{ fresh} & \\
\lceil \mathsf{x} \mathbin{⨾} \tau \rceil & = & \mathsf{x} \mathbin{⨾} \tau & \\
\lceil \mathtt{bind}\,\mathsf{x} = a\ \mathtt{in}\,\kappa \rceil & = & \lceil \kappa \rceil\{\lceil a \rceil/\mathsf{x}\} & \\
\lceil \mathsf{x} \rceil & = & \mathsf{x} &
\end{array}$$

E.2 From source language expressions to canonical forms

Any expression e of the source language can be transformed into a canonical form whose unwinding is e. Let Δ denote a possibly empty list of mappings from binding variables to atoms. We note these lists extensionally by separating elements by a semicolon, that is, $\mathsf{x}_1{\mapsto}a_1; \ldots; \mathsf{x}_n{\mapsto}a_n$ and use ε to denote the empty list. We define an operation $\mathsf{term}(\Delta, \mathsf{x})$ which takes a list of mappings Δ and a binding variable x and constructs the canonical form whose bindings are those listed in Δ and whose body is x, that is:

$$\mathsf{term}(\varepsilon, \mathsf{x}) \stackrel{\text{def}}{=} \mathsf{x}$$
$$\mathsf{term}((\mathsf{y}{\mapsto}a; \Delta), \mathsf{x}) \stackrel{\text{def}}{=} \mathsf{bind}\,\mathsf{y} = a\,\mathsf{in}\,\mathsf{term}(\Delta, \mathsf{x})$$

We can now define the function $[\![e]\!]$ that transforms an expression e into a pair (Δ, x) formed by a list of mappings Δ and a binding variable x that will be bound to the atom representing e. The definition is as follows, where $\mathsf{x}_\circ$ is a fresh binding variable.

$$\begin{aligned}
[\![c]\!] &= ((\mathsf{x}_\circ{\mapsto}c), \mathsf{x}_\circ) \\
[\![x]\!] &= ((\mathsf{x}_\circ{\mapsto}x), \mathsf{x}_\circ) \\
[\![\lambda x.e]\!] &= ((\mathsf{x}_\circ{\mapsto}\lambda x.\mathsf{term}[\![e]\!]), \mathsf{x}_\circ) \\
[\![\pi_i e]\!] &= ((\Delta; \mathsf{x}_\circ{\mapsto}\pi_i \mathsf{x}), \mathsf{x}_\circ) && \text{where } (\Delta, \mathsf{x}) = [\![e]\!] \\
[\![e_1 e_2]\!] &= ((\Delta_1; \Delta_2; \mathsf{x}_\circ{\mapsto}\mathsf{x}_1\mathsf{x}_2), \mathsf{x}_\circ) && \text{where } (\Delta_1, \mathsf{x}_1) = [\![e_1]\!],\ (\Delta_2, \mathsf{x}_2) = [\![e_2]\!] \\
[\![(e_1, e_2)]\!] &= ((\Delta_1; \Delta_2; \mathsf{x}_\circ{\mapsto}(\mathsf{x}_1, \mathsf{x}_2)), \mathsf{x}_\circ) && \text{where } (\Delta_1, \mathsf{x}_1) = [\![e_1]\!],\ (\Delta_2, \mathsf{x}_2) = [\![e_2]\!] \\
[\![(e{\in}\tau)\ ?\ e_1 : e_2]\!] &= ((\Delta; \Delta_1; \Delta_2; \mathsf{x}_\circ{\mapsto}(\mathsf{x}{\in}\tau)\ ?\ \mathsf{x}_1 : \mathsf{x}_2), \mathsf{x}_\circ) \\
& && \text{where } (\Delta, \mathsf{x}) = [\![e]\!],\ (\Delta_1, \mathsf{x}_1) = [\![e_1]\!],\ (\Delta_2, \mathsf{x}_2) = [\![e_2]\!] \\
[\![\mathsf{let}\,x = e_1\,\mathsf{in}\,e_2\,]\!] &= (\Delta_1; \Delta_2; \mathsf{x}_\circ \mapsto \mathsf{let}\,\mathsf{x}_1\,\mathsf{in}\,\mathsf{x}_2\,, \mathsf{x}_\circ) \\
& && \text{where } (\Delta_1, \mathsf{x}_1) = [\![e_1]\!], (\Delta_2, \mathsf{x}_2) = [\![e_2\{\mathsf{x}_1/x\}]\!] \\
[\![e \mathbin{⨾} \tau]\!] &= (\Delta; \mathsf{x}_\circ \mapsto \mathsf{x} \mathbin{⨾} \tau, \mathsf{x}_\circ) && \text{where } (\Delta, \mathsf{x}) = [\![e]\!]
\end{aligned}$$

It is easy to see that, for any term of the source language e, $\lceil \mathsf{term}([\![e]\!]) \rceil = e$.

E.3 From canonical forms to a MSC form

It is easy to transform a canonical form into a MSC-form that has the same unwinding. This can be done by applying the rewriting rules below, that are confluent and normalizing.

$$\begin{array}{lcll}
\begin{array}{l}\mathsf{bind}\,\mathsf{x}_1 = a_1\,\mathsf{in} \\ \mathsf{bind}\,\mathsf{x}_2 = a_2\,\mathsf{in}\,\kappa\end{array} & \dashrightarrow & \mathsf{bind}\,\mathsf{x}_1 = a_1\,\mathsf{in}\,\kappa\{\mathsf{x}_1/\mathsf{x}_2\} & a_1 \equiv_\kappa a_2 \qquad (35) \\
\mathsf{bind}\,\mathsf{x} = a\,\mathsf{in}\,\kappa & \dashrightarrow & \kappa & \mathsf{x} \notin \mathsf{fv}(\kappa) \qquad (36) \\
\begin{array}{l}\mathsf{bind}\,\mathsf{x} = \lambda y.(\\ \quad \mathsf{bind}\,\mathsf{z} = a\,\mathsf{in}\,\kappa_\circ\,) \\ \mathsf{in}\,\kappa\end{array} & \dashrightarrow & \begin{array}{l}\mathsf{bind}\,\mathsf{z} = a\,\mathsf{in} \\ \mathsf{bind}\,\mathsf{x} = \lambda y.\kappa_\circ\,\mathsf{in}\,\kappa\end{array} & y \notin \mathsf{fv}(a), \mathsf{z} \notin \mathsf{fv}(\kappa) \qquad (37)
\end{array}$$

$$\kappa_1 \dashrightarrow \kappa_2 \qquad\qquad \exists \kappa_1'.\ \kappa_1 \equiv_\kappa \kappa_1' \dashrightarrow \kappa_2 \qquad (38)$$

Rule (35) implements the maximal sharing: if two variables bind atoms with the same unwinding (modulo α-conversion), then the variables are unified. Rule (36) removes useless bindings. Rule (37) extrudes bindings from abstractions of variables that do not occur in the argument of the binding. Rule (38) applies the previous rule modulo the canonical equivalence: in practice it applies the swap of binding defined in Definition 3.1 as many times as it is needed to apply one of the other rules. As customary, these rules can be applied under any context.

The transformation above transforms every canonical form into an MSC-form that has the same unwinding. It thus allows to compute $\mathsf{MSC}(e)$ for any expression e of the source language.

F TYPE OPERATORS

The algorithmic type system presented in this work use the following type-operators:

$$\begin{aligned}
\mathsf{dom}(t) &= \max\{u \mid t \leq u \to \mathbb{1}\} \\
t \circ s &= \min\{u \mid t \leq s \to u\} \\
\boldsymbol{\pi}_1(t) &= \min\{u \mid t \leq u \times \mathbb{1}\} \\
\boldsymbol{\pi}_2(t) &= \min\{u \mid t \leq \mathbb{1} \times u\}
\end{aligned}$$

In words, $t \circ s$ is the best (i.e., smallest wrt $\leq$) type we can deduce for the application of a function of type t to an argument of type s. Projection and domain are standard. All these operators can be effectively computed as shown below (see Castagna et al. [2022a]; Frisch et al. [2008] for details and proofs).

For $t \overset{\text{DNF}}{\simeq} \bigvee_{i\in I}\left(\bigwedge_{p'\in P'_i} \alpha_{p'} \wedge \bigwedge_{n'\in N'_i} \neg\alpha'_{n'} \wedge \bigwedge_{p\in P_i}(s_p \to t_p) \wedge \bigwedge_{n\in N_i} \neg(s'_n \to t'_n)\right)$, the first two operators are computed by:

$$\begin{aligned}
\mathsf{dom}(t) &= \bigwedge_{i\in I} \bigvee_{p\in P_i} s_p \\
t \circ s &= \bigvee_{i\in I}\left(\bigvee_{\{Q\subsetneq P_i \mid s \not\leq \bigvee_{q\in Q} s_q\}}\left(\bigwedge_{p\in P_i\setminus Q} t_p\right)\right) \qquad (\text{for } s \leq \mathsf{dom}(t))
\end{aligned}$$

For $t \overset{\text{DNF}}{\simeq} \bigvee_{i\in I}\left(\bigwedge_{p'\in P'_i} \alpha_{p'} \wedge \bigwedge_{n'\in N'_i} \neg\alpha'_{n'} \wedge \bigwedge_{p\in P_i}(s_p, t_p) \wedge \bigwedge_{n\in N_i} \neg(s'_n, t'_n)\right)$ the last two operators are computed by

$$\begin{aligned}
\boldsymbol{\pi}_1(t) &= \bigvee_{i\in I} \bigvee_{N'\subseteq N_i}\left(\bigwedge_{p\in P_i} s_p \wedge \bigwedge_{n\in N'} \neg s'_n\right) \\
\boldsymbol{\pi}_2(t) &= \bigvee_{i\in I} \bigvee_{N'\subseteq N_i}\left(\bigwedge_{p\in P_i} t_p \wedge \bigwedge_{n\in N'} \neg t'_n\right)
\end{aligned}$$

G ALGORITHMIC TYPE SYSTEM

$$\begin{array}{llll}
\textbf{Atom annots} & \mathbb{a} & ::= & \varnothing \mid \lambda(\mathbf{u}, \mathbb{k}) \mid (\rho, \rho) \mid @(\Sigma, \Sigma) \mid \pi(\Sigma) \mid \mathbb{0}(\Sigma) \mid \in_1(\Sigma) \mid \in_2(\Sigma) \mid \bigwedge(\{\mathbb{a}, \ldots, \mathbb{a}\}) \\
\textbf{Form annots} & \mathbb{k} & ::= & \rho \mid \mathsf{keep}\ (\mathbb{a}, \{(\mathbf{u}, \mathbb{k}), \ldots, (\mathbf{u}, \mathbb{k})\}) \mid \mathsf{skip}\ \mathbb{k} \mid \bigwedge(\{\mathbb{k}, \ldots, \mathbb{k}\})
\end{array}$$

The algorithmic type system is defined by the following deduction rules:

$$[\textsc{Const-Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [c \mid \varnothing] : b_c} \qquad [\textsc{Ax-Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [x \mid \varnothing] : \Gamma(x)}$$

$$[{\to}\text{I-}\textsc{Alg}]\ \frac{\Gamma, x : \mathbf{u} \vdash_{\mathcal{A}} [\kappa \mid \Bbbk] : t}{\Gamma \vdash_{\mathcal{A}} [\lambda x.\kappa \mid \lambda(\mathbf{u}, \Bbbk)] : \mathbf{u} \to t}$$

$$[{\to}\text{E-}\textsc{Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [\mathsf{x}_1\mathsf{x}_2 \mid @(\Sigma_1, \Sigma_2)] : t_1 \circ t_2}\ \begin{array}{l} t_1 = \Gamma(\mathsf{x}_1)\Sigma_1,\ \ t_2 = \Gamma(\mathsf{x}_2)\Sigma_2 \\ t_1 \leq \mathbb{0} \to \mathbb{1},\ \ t_2 \leq \mathsf{dom}(t_1) \end{array}$$

$$[{\times}\text{I-}\textsc{Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [(\mathsf{x}_1, \mathsf{x}_2) \mid (\rho_1, \rho_2)] : t_1 \times t_2}\ t_1 = \Gamma(\mathsf{x}_1)\rho_1,\ \ t_2 = \Gamma(\mathsf{x}_2)\rho_2$$

$$[{\times}\text{E}_1\text{-}\textsc{Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [\pi_1\mathsf{x} \mid \pi(\Sigma)] : \boldsymbol{\pi}_1(t)}\ \begin{array}{l} t = \Gamma(\mathsf{x})\Sigma \\ t \leq (\mathbb{1} \times \mathbb{1}) \end{array} \qquad [{\times}\text{E}_2\text{-}\textsc{Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [\pi_2\mathsf{x} \mid \pi(\Sigma)] : \boldsymbol{\pi}_2(t)}\ \begin{array}{l} t = \Gamma(\mathsf{x})\Sigma \\ t \leq (\mathbb{1} \times \mathbb{1}) \end{array}$$

$$[\mathbb{0}\text{-}\textsc{Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [(\mathsf{x}{\in}\tau)\,?\,\mathsf{x}_1 : \mathsf{x}_2 \mid \mathbb{0}(\Sigma)] : \mathbb{0}}\ \Gamma(\mathsf{x})\Sigma \simeq \mathbb{0}$$

$$[\in_1\text{-}\textsc{Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [(\mathsf{x}{\in}\tau)\,?\,\mathsf{x}_1 : \mathsf{x}_2 \mid \in_1(\Sigma)] : \Gamma(\mathsf{x}_1)}\ \Gamma(\mathsf{x})\Sigma \leq \tau$$

$$[\in_2\text{-}\textsc{Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [(\mathsf{x}{\in}\tau)\,?\,\mathsf{x}_1 : \mathsf{x}_2 \mid \in_2(\Sigma)] : \Gamma(\mathsf{x}_2)}\ \Gamma(\mathsf{x})\Sigma \leq \neg\tau$$

$$[\wedge\text{-}\textsc{Alg}]\ \frac{(\forall i \in I) \quad \Gamma \vdash_{\mathcal{A}} [a \mid \mathbb{a}_i] : t_i}{\Gamma \vdash_{\mathcal{A}} [a \mid \bigwedge(\{\mathbb{a}_i\}_{i \in I})] : \bigwedge_{i \in I} t_i}\ I \neq \varnothing$$

$$[\textsc{Var-Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [\mathsf{x} \mid \rho] : \Gamma(\mathsf{x})\rho} \qquad [\textsc{Bind}_1\text{-}\textsc{Alg}]\ \frac{\Gamma \vdash_{\mathcal{A}} [\kappa \mid \Bbbk] : t}{\Gamma \vdash_{\mathcal{A}} [\mathsf{bind}\,\mathsf{x} = a\ \mathsf{in}\ \kappa \mid \mathsf{skip}\ \Bbbk] : t}\ \mathsf{x} \notin \mathsf{dom}(\Gamma)$$

$$[\textsc{Bind}_2\text{-}\textsc{Alg}]\ \frac{\Gamma \vdash_{\mathcal{A}} [a \mid \mathbb{a}] : s \qquad (\forall i \in I) \quad \Gamma, \mathsf{x} : s \wedge \mathbf{u}_i \vdash_{\mathcal{A}} [\kappa \mid \Bbbk_i] : t_i}{\Gamma \vdash_{\mathcal{A}} [\mathsf{bind}\,\mathsf{x} = a\ \mathsf{in}\ \kappa \mid \mathsf{keep}\ (\mathbb{a}, \{(\mathbf{u}_i, \Bbbk_i)\}_{i \in I})] : \bigvee_{i \in I} t_i}\ I \neq \varnothing,\ \ \bigvee_{i \in I} \mathbf{u}_i \simeq \mathbb{1}$$

$$[\wedge\text{-}\textsc{Alg}]\ \frac{(\forall i \in I) \quad \Gamma \vdash_{\mathcal{A}} [\kappa \mid \Bbbk_i] : t_i}{\Gamma \vdash_{\mathcal{A}} [\kappa \mid \bigwedge(\{\Bbbk_i\}_{i \in I})] : \bigwedge_{i \in I} t_i}\ I \neq \varnothing$$

To extend the system to type the extensions presented in Appendix A the following rules must be added:

$$[\textsc{Let-Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [\mathsf{let}\,\mathsf{x}_1\ \mathsf{in}\,\mathsf{x}_2 \mid \varnothing] : \Gamma(\mathsf{x}_2)}\ \mathsf{x}_1 \in \mathsf{dom}(\Gamma)$$

$$[\textsc{Constr-Alg}]\ \frac{}{\Gamma \vdash_{\mathcal{A}} [\mathsf{x} \between \tau \mid \between(\Sigma)] : \Gamma(\mathsf{x})}\ \Gamma(\mathsf{x})\Sigma \leq \tau$$

H FULL RECONSTRUCTION SYSTEM

H.1 Monomorphic reconstruction

Split annotations	$\mathcal{S}$	::=	$\{(\mathbf{u},\mathcal{K}),\dots,(\mathbf{u},\mathcal{K})\}$
Atoms intermediate annot.	$\mathcal{A}$	::=	$\mathtt{infer} \mid \mathtt{untyp} \mid \mathtt{typ} \mid \bigwedge(\{\mathcal{A},\dots,\mathcal{A}\},\{\mathcal{A},\dots,\mathcal{A}\})$ $\mid \in_1 \mid \in_2 \mid \lambda(\mathbf{u},\mathcal{K})$
Forms intermediate annot.	$\mathcal{K}$	::=	$\mathtt{infer} \mid \mathtt{untyp} \mid \mathtt{typ} \mid \bigwedge(\{\mathcal{K},\dots,\mathcal{K}\},\{\mathcal{K},\dots,\mathcal{K}\})$ $\mid \mathtt{try\text{-}skip}\ (\mathcal{K}) \mid \mathtt{try\text{-}keep}\ (\mathcal{A},\mathcal{K},\mathcal{K})$ $\mid \mathtt{propagate}\ (\mathcal{A},\mathbb{\Gamma},\mathcal{S},\mathcal{S}) \mid \mathtt{skip}\ (\mathcal{K}) \mid \mathtt{keep}\ (\mathcal{A},\mathcal{S},\mathcal{S})$

H.1.1 Deduction rules for $\vdash_{\mathcal{R}}$ judgements.

$$[\textsc{Ok}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \mathtt{typ}\rangle \Rightarrow \mathtt{Ok}(\mathtt{typ})} \qquad [\textsc{Fail}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \mathtt{untyp}\rangle \Rightarrow \mathtt{Fail}}$$

$$[\textsc{Const}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle c \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Ok}(\mathtt{typ})}$$

$$[\textsc{AxOk}]\ \frac{x \in \mathrm{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{R}} \langle x \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Ok}(\mathtt{typ})} \qquad [\textsc{AxFail}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle x \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Fail}}$$

$$[\textsc{PairVar}_i]\ \frac{\mathrm{x}_i \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{R}} \langle (\mathrm{x}_1,\mathrm{x}_2) \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Var}\ (\mathrm{x}_i, \mathtt{infer}, \mathtt{untyp})}$$

$$[\textsc{PairOk}]\ \frac{\{\mathrm{x}_1,\mathrm{x}_2\} \subseteq \mathrm{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{R}} \langle (\mathrm{x}_1,\mathrm{x}_2) \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Ok}(\mathtt{typ})}$$

$$[\textsc{ProjVar}]\ \frac{\mathrm{x} \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{R}} \langle \pi_i \mathrm{x} \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Var}\ (\mathrm{x}, \mathtt{infer}, \mathtt{untyp})}$$

$$[\textsc{ProjInfer}]\ \frac{\Psi = \mathtt{tally\_infer}(\{\Gamma(\mathrm{x}) \dot{\leq} \alpha \times \beta\})}{\Gamma \vdash_{\mathcal{R}} \langle \pi_i \mathrm{x} \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Subst}(\Psi, \mathtt{typ}, \mathtt{untyp})}\ \alpha,\beta \in \mathcal{V}_P \text{ fresh}$$

$$[\textsc{AppVar}_i]\ \frac{\mathrm{x}_i \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{R}} \langle \mathrm{x}_1\mathrm{x}_2 \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Var}\ (\mathrm{x}_i, \mathtt{infer}, \mathtt{untyp})}$$

$$[\textsc{AppInfer}]\ \frac{\Psi = \mathtt{tally\_infer}(\{\Gamma(\mathrm{x}_1) \dot{\leq} \Gamma(\mathrm{x}_2) \to \alpha\})}{\Gamma \vdash_{\mathcal{R}} \langle \mathrm{x}_1\mathrm{x}_2 \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Subst}(\Psi, \mathtt{typ}, \mathtt{untyp})}\ \alpha \in \mathcal{V}_P \text{ fresh}$$

$$[\textsc{CaseVar}]\ \frac{\mathrm{x} \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{R}} \langle (\mathrm{x} \in \tau)\,?\,\mathrm{x}_1 : \mathrm{x}_2 \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Var}\ (\mathrm{x}, \mathtt{infer}, \mathtt{untyp})}$$

$$[\textsc{CaseSplit}]\ \frac{\Gamma(\mathrm{x}) \not\leq \tau \qquad \Gamma(\mathrm{x}) \not\leq \neg\tau}{\Gamma \vdash_{\mathcal{R}} \langle (\mathrm{x} \in \tau)\,?\,\mathrm{x}_1 : \mathrm{x}_2 \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Split}(\{(\mathrm{x} : \tau)\}, \mathtt{infer}, \mathtt{infer})}$$

$$[\textsc{CaseThen}]\ \frac{\Gamma(\mathtt{x}) \leq \tau \qquad \Psi = \mathtt{tally\_infer}(\{\Gamma(\mathtt{x}) \dot{\leq} \mathbb{0}\})}{\Gamma \vdash_{\mathcal{R}} \langle (\mathtt{x}{\in}\tau)\,?\,\mathtt{x}_1 : \mathtt{x}_2 \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Subst}(\Psi, \mathtt{typ}, \in_1)}$$

$$[\textsc{CaseElse}]\ \frac{\Gamma(\mathtt{x}) \leq \neg\tau \qquad \Psi = \mathtt{tally\_infer}(\{\Gamma(\mathtt{x}) \dot{\leq} \mathbb{0}\})}{\Gamma \vdash_{\mathcal{R}} \langle (\mathtt{x}{\in}\tau)\,?\,\mathtt{x}_1 : \mathtt{x}_2 \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Subst}(\Psi, \mathtt{typ}, \in_2)}$$

$$[\textsc{CaseVar}_i]\ \frac{\mathtt{x}_i \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{R}} \langle (\mathtt{x}{\in}\tau)\,?\,\mathtt{x}_1 : \mathtt{x}_2 \mid \in_i\rangle \Rightarrow \mathtt{Var}\ (\mathtt{x}_i, \mathtt{typ}, \mathtt{untyp})}$$

$$[\textsc{CaseOk}_i]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle (\mathtt{x}{\in}\tau)\,?\,\mathtt{x}_1 : \mathtt{x}_2 \mid \in_i\rangle \Rightarrow \mathtt{Ok}(\mathtt{typ})}$$

$$[\textsc{LambdaInfer}]\ \frac{\Gamma \vdash_{\mathcal{R}} \langle \lambda x.\kappa \mid \lambda(\boldsymbol{\alpha}, \mathtt{infer})\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \lambda x.\kappa \mid \mathtt{infer}\rangle \Rightarrow \mathbb{R}}\ \boldsymbol{\alpha} \in \mathcal{V}_M \text{ fresh}$$

$$[\textsc{LambdaEmpty}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle \lambda x.\kappa \mid \lambda(\mathbb{0}, \mathcal{K})\rangle \Rightarrow \mathtt{Fail}}$$

$$[\textsc{Lambda}]\ \frac{\Gamma, x : \mathbf{u} \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K}\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \lambda x.\kappa \mid \lambda(\mathbf{u}, \mathcal{K})\rangle \Rightarrow \mathtt{map}(X \mapsto \lambda(\mathbf{u}, X),\ \mathbb{R})}$$

with $\mathtt{map}(X \mapsto f(X),\ \mathbb{R})$ an auxiliary function that applies f to each intermediate annotation in $\mathbb{R}$:

$$\begin{aligned}
\mathtt{map}(X \mapsto f(X),\ \mathtt{Ok}(\mathcal{H})) &\overset{\text{def}}{=} \mathtt{Ok}(f(\mathcal{H}))\\
\mathtt{map}(X \mapsto f(X),\ \mathtt{Fail}) &\overset{\text{def}}{=} \mathtt{Fail}\\
\mathtt{map}(X \mapsto f(X),\ \mathtt{Split}(\Gamma, \mathcal{H}_1, \mathcal{H}_2)) &\overset{\text{def}}{=} \mathtt{Split}(\Gamma, f(\mathcal{H}_1), f(\mathcal{H}_2))\\
\mathtt{map}(X \mapsto f(X),\ \mathtt{Subst}(\Psi, \mathcal{H}_1, \mathcal{H}_2)) &\overset{\text{def}}{=} \mathtt{Subst}(\Psi, f(\mathcal{H}_1), f(\mathcal{H}_2))\\
\mathtt{map}(X \mapsto f(X),\ \mathtt{Var}\ (\mathtt{x}, \mathcal{H}_1, \mathcal{H}_2)) &\overset{\text{def}}{=} \mathtt{Var}\ (\mathtt{x}, f(\mathcal{H}_1), f(\mathcal{H}_2))
\end{aligned}$$

$$[\textsc{FormVar}]\ \frac{\mathtt{x} \notin \mathrm{dom}(\Gamma)}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{x} \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Var}\ (\mathtt{x}, \mathtt{infer}, \mathtt{untyp})}$$

$$[\textsc{FormOk}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{x} \mid \mathtt{infer}\rangle \Rightarrow \mathtt{Ok}(\mathtt{typ})}$$

$$[\textsc{BindInfer}]\ \frac{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathtt{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{try\text{-}skip}\ (\mathtt{infer})\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathtt{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{infer}\rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{BindTrySkip}_1]\ \frac{\begin{array}{c}\Gamma \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathtt{Var}\ (\mathrm{x}, \mathcal{K}_1, \mathcal{K}_2) \\ \Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{try\text{-}keep}\ (\mathtt{infer}, \mathcal{K}_1, \mathcal{K}_2) \rangle \Rightarrow \mathbb{R}\end{array}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{try\text{-}skip}\ (\mathcal{K}) \rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{BindTrySkip}_2]\ \frac{\Gamma \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathtt{Ok}(\mathcal{K}')}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{try\text{-}skip}\ (\mathcal{K}) \rangle \Rightarrow \mathtt{Ok}(\mathtt{skip}\ (\mathcal{K}'))}$$

$$[\textsc{BindTrySkip}_3]\ \frac{\Gamma \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{try\text{-}skip}\ (\mathcal{K}) \rangle \Rightarrow \mathtt{map}(X \mapsto \mathtt{try\text{-}skip}\ (X),\ \mathbb{R})}$$

$$[\textsc{BindSkip}_1]\ \frac{\Gamma \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathtt{Var}\ (\mathrm{x}, \mathcal{K}_1, \mathcal{K}_2) \qquad \Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{skip}\ (\mathcal{K}_2) \rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{skip}\ (\mathcal{K}) \rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{BindSkip}_2]\ \frac{\Gamma \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{skip}\ (\mathcal{K}) \rangle \Rightarrow \mathtt{map}(X \mapsto \mathtt{skip}\ (X),\ \mathbb{R})}$$

$$[\textsc{BindTryKeep}_1]\ \frac{\begin{array}{c}\Gamma \vdash^*_{\mathcal{R}} \langle a \mid \mathcal{A} \rangle \Rightarrow \mathtt{Ok}(\mathcal{A}') \\ \Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{keep}\ (\mathcal{A}', \{(\mathbb{1}, \mathcal{K}_1)\}, \varnothing) \rangle \Rightarrow \mathbb{R}\end{array}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{try\text{-}keep}\ (\mathcal{A}, \mathcal{K}_1, \mathcal{K}_2) \rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{BindTryKeep}_2]\ \frac{\Gamma \vdash^*_{\mathcal{R}} \langle a \mid \mathcal{A} \rangle \Rightarrow \mathtt{Fail} \qquad \Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{skip}\ (\mathcal{K}_2) \rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{try\text{-}keep}\ (\mathcal{A}, \mathcal{K}_1, \mathcal{K}_2) \rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{BindTryKeep}_3]\ \frac{\Gamma \vdash^*_{\mathcal{R}} \langle a \mid \mathcal{A} \rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{try\text{-}keep}\ (\mathcal{A}, \mathcal{K}_1, \mathcal{K}_2) \rangle \Rightarrow \mathbb{R}'}$$

where $\mathbb{R}' = \mathtt{map}(X \mapsto \mathtt{try\text{-}keep}\ (X, \mathcal{K}_1, \mathcal{K}_2),\ \mathbb{R})$.

$$[\textsc{BindOk}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{keep}\ (\mathcal{A}, \varnothing, \mathcal{S}) \rangle \Rightarrow \mathtt{Ok}(\mathtt{keep}\ (\mathcal{A}, \varnothing, \mathcal{S}))}$$

$$[\textsc{BindSplit}_1]\ \frac{\begin{array}{c}\Gamma \vdash_{\mathcal{P}} \langle a \mid \mathcal{A} \rangle \Rightarrow \mathbb{a} \qquad \Gamma \vdash_{\mathcal{A}} [a \mid \mathbb{a}] : s \qquad \Gamma, \mathrm{x} : s \wedge \mathbf{u} \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathtt{Ok}(\mathcal{K}') \\ \Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{keep}\ (\mathcal{A}, \mathcal{S}, \{(\mathbf{u}, \mathcal{K}')\} \cup \mathcal{S}') \rangle \Rightarrow \mathbb{R}\end{array}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{keep}\ (\mathcal{A}, \{(\mathbf{u}, \mathcal{K})\} \cup \mathcal{S}, \mathcal{S}') \rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{BindSplit}_2]\ \frac{\begin{array}{c}\Gamma \vdash_{\mathcal{P}} \langle a \mid \mathcal{A} \rangle \Rightarrow \mathbb{a} \\ \Gamma \vdash_{\mathcal{A}} [a \mid \mathbb{a}] : s \qquad \Gamma, \mathrm{x} : s \wedge \mathbf{u} \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathtt{Split}(\Gamma', \mathcal{K}_1, \mathcal{K}_2) \\ \mathrm{x} \in \mathrm{dom}(\Gamma') \qquad \Gamma \vdash_{\mathcal{E}} (a : \neg(\mathbf{u} \wedge \Gamma'(\mathrm{x}))) \Rightarrow \mathbb{\Gamma}_1 \qquad \Gamma \vdash_{\mathcal{E}} (a : \neg(\mathbf{u} \setminus \Gamma'(\mathrm{x}))) \Rightarrow \mathbb{\Gamma}_2\end{array}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa \mid \mathtt{keep}\ (\mathcal{A}, \{(\mathbf{u}, \mathcal{K})\} \cup \mathcal{S}, \mathcal{S}') \rangle \Rightarrow \mathtt{Split}(\Gamma' \setminus \mathrm{x}, \mathcal{K}_1', \mathcal{K}_2')}$$

where:

- $\mathcal{K}_1' = \mathtt{propagate}\ (\mathcal{A}, \mathbb{\Gamma}_1 \cup \mathbb{\Gamma}_2, \{(\mathbf{u} \wedge \Gamma'(\mathrm{x}), \mathcal{K}_1),\ (\mathbf{u} \setminus \Gamma'(\mathrm{x}), \mathcal{K}_2)\} \cup \mathcal{S}, \mathcal{S}')$
- $\mathcal{K}_2' = \mathtt{keep}\ (\mathcal{A}, \{(\mathbf{u}, \mathcal{K}_2)\} \cup \mathcal{S}, \mathcal{S}')$

$$[\textsc{BindSplit}_3]\ \frac{\Gamma \vdash_{\mathcal{P}} \langle a \mid \mathcal{A}\rangle \Rightarrow \mathbb{a} \qquad \Gamma \vdash_{\mathcal{A}} [a \mid \mathbb{a}] : s \qquad \Gamma, \mathrm{x} : s \wedge \mathbf{u} \vdash^{*}_{\mathcal{R}} \langle \kappa \mid \mathcal{K}\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\,\mathtt{in}\,\kappa \mid \mathtt{keep}\ (\mathcal{A}, \{(\mathbf{u}, \mathcal{K})\} \cup \mathcal{S}, \mathcal{S}')\rangle \Rightarrow \mathbb{R}'}$$

where $\mathbb{R}' = \mathtt{map}(X \mapsto \mathtt{keep}\ (\mathcal{A}, \{(\mathbf{u}, X)\} \cup \mathcal{S}, \mathcal{S}'),\ \mathbb{R})$.

$$[\textsc{BindProp}_1]\ \frac{\Gamma' \in \mathbb{\Gamma} \qquad \mathtt{compatible}(\Gamma, \Gamma')}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\,\mathtt{in}\,\kappa \mid \mathtt{propagate}\ (\mathcal{A}, \mathbb{\Gamma}, \mathcal{S}, \mathcal{S}')\rangle \Rightarrow \mathtt{Split}(\Gamma'', \mathcal{K}', \mathcal{K}')}$$

where:

- $\mathtt{compatible}(\Gamma, \Gamma') \Leftrightarrow (\mathrm{dom}(\Gamma') \subseteq \mathrm{dom}(\Gamma)) \wedge (\forall \mathrm{x} \in \mathrm{dom}(\Gamma').\ (\Gamma(\mathrm{x}) \wedge \Gamma'(\mathrm{x}) \not\simeq \mathbb{0}) \vee (\Gamma(\mathrm{x}) \simeq \mathbb{0}))$
- $\mathcal{K}' = \mathtt{propagate}\ (\mathcal{A}, \mathbb{\Gamma} \setminus \{\Gamma'\}, \mathcal{S}, \mathcal{S}')$
- $\Gamma'' = \{(\mathrm{x} : \mathbf{u}) \in \Gamma' \mid \Gamma(\mathrm{x}) \not\leq \mathbf{u}\}$

$$[\textsc{BindProp}_2]\ \frac{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\,\mathtt{in}\,\kappa \mid \mathtt{keep}\ (\mathcal{A}, \mathcal{S}, \mathcal{S}')\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \mathtt{bind}\,\mathrm{x} = a\,\mathtt{in}\,\kappa \mid \mathtt{propagate}\ (\mathcal{A}, \mathbb{\Gamma}, \mathcal{S}, \mathcal{S}')\rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{InterEmpty}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \ \bigwedge(\varnothing, \varnothing)\rangle \Rightarrow \mathtt{Fail}}$$

$$[\textsc{InterOk}]\ \frac{}{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \ \bigwedge(\varnothing, S)\rangle \Rightarrow \mathtt{Ok}(\bigwedge(\varnothing, S))}$$

$$[\textsc{Inter}_1]\ \frac{\Gamma \vdash^{*}_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathtt{Ok}(\mathcal{H}') \qquad \Gamma \vdash_{\mathcal{R}} \langle \eta \mid \ \bigwedge(S, \{\mathcal{H}'\} \cup S')\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \ \bigwedge(\{\mathcal{H}\} \cup S, S')\rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{Inter}_2]\ \frac{\Gamma \vdash^{*}_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathtt{Fail} \qquad \Gamma \vdash_{\mathcal{R}} \langle \eta \mid \ \bigwedge(S, S')\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \ \bigwedge(\{\mathcal{H}\} \cup S, S')\rangle \Rightarrow \mathbb{R}}$$

$$[\textsc{Inter}_3]\ \frac{\Gamma \vdash^{*}_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \ \bigwedge(\{\mathcal{H}\} \cup S, S')\rangle \Rightarrow \mathtt{map}(X \mapsto (\bigwedge(\{X\} \cup S, S')),\ \mathbb{R})}$$

H.1.2 Deduction rules for $\vdash^{*}_{\mathcal{R}}$ judgements.

$$[\textsc{Iterate}_1]\ \frac{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathtt{Split}(\Gamma', \mathcal{H}_1, \mathcal{H}_2) \qquad \Gamma \vdash^{*}_{\mathcal{R}} \langle \eta \mid \mathcal{H}_1\rangle \Rightarrow \mathbb{R}'}{\Gamma \vdash^{*}_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathbb{R}'}\ \Gamma' = \varnothing$$

$$[\textsc{Iterate}_2]\ \frac{\begin{array}{c}\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathtt{Subst}(\{\psi_i\}_{i \in I}, \mathcal{H}_1, \mathcal{H}_2) \\ \Gamma \vdash^{*}_{\mathcal{R}} \langle \eta \mid \ \bigwedge(\{\mathtt{fresh}(\mathcal{H}_1\psi_i)\}_{i \in I} \cup \{\mathcal{H}_2\}, \varnothing)\rangle \Rightarrow \mathbb{R}'\end{array}}{\Gamma \vdash^{*}_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathbb{R}'}\ \forall i \in I.\ \psi_i \# \Gamma$$

$$[\textsc{Stop}]\ \frac{\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathbb{R}}{\Gamma \vdash^{*}_{\mathcal{R}} \langle \eta \mid \mathcal{H}\rangle \Rightarrow \mathbb{R}}$$

with $\mathcal{H}\psi$ denoting the intermediate annotation $\mathcal{H}$ in which the substitution ψ has been applied recursively to every type (in lambdas and binding annotations), and $\mathtt{fresh}(\mathcal{H})$ denotes $\mathcal{H}$ where all the monomorphic type variables not in Γ have been substituted by fresh ones (in order to decorrelate the different branches).

The following rules can be added to support the extensions presented in Appendix A:

$$[\textsc{LetOk}]\ \frac{\{\mathtt{x}_1,\mathtt{x}_2\}\subseteq\mathrm{dom}(\Gamma)}{\Gamma\vdash_{\mathcal{R}}\langle\mathtt{let}\,\mathtt{x}_1\,\mathtt{in}\,\mathtt{x}_2\mid\mathtt{infer}\rangle\Rightarrow\mathtt{Ok}(\mathtt{typ})}$$

$$[\textsc{LetVar}_i]\ \frac{\mathtt{x}_i\notin\mathrm{dom}(\Gamma)}{\Gamma\vdash_{\mathcal{R}}\langle\mathtt{let}\,\mathtt{x}_1\,\mathtt{in}\,\mathtt{x}_2\mid\mathtt{infer}\rangle\Rightarrow\mathtt{Var}\ (\mathtt{x}_i,\mathtt{infer},\mathtt{untyp})}$$

$$[\textsc{ConstrInfer}]\ \frac{\Sigma=\mathsf{tally\_infer}(\{\Gamma(\mathtt{x})\ \dot{\leq}\ \tau\})}{\Gamma\vdash_{\mathcal{R}}\langle\mathtt{x}\mathbin{⨾}\tau\mid\mathtt{infer}\rangle\Rightarrow\mathtt{Subst}(\Sigma,\mathtt{typ},\mathtt{untyp})}$$

$$[\textsc{ConstrVar}]\ \frac{}{\Gamma\vdash_{\mathcal{R}}\langle\mathtt{x}\mathbin{⨾}\tau\mid\mathtt{infer}\rangle\Rightarrow\mathtt{Var}\ (\mathtt{x},\mathtt{infer},\mathtt{untyp})}$$

H.2 Polymorphic reconstruction

In the following, $\mathtt{refresh}(t)$ denotes a renaming from $\mathrm{vars}(t)\cap\mathcal{V}_P$ to fresh polymorphic variables.

$$[\textsc{Const}]\ \frac{}{\Gamma\vdash_{\mathcal{P}}\langle c\mid\mathtt{typ}\rangle\Rightarrow\varnothing}\qquad[\textsc{Ax}]\ \frac{}{\Gamma\vdash_{\mathcal{P}}\langle x\mid\mathtt{typ}\rangle\Rightarrow\varnothing}\ x\in\mathrm{dom}(\Gamma)$$

$$[\textsc{Pair}]\ \frac{\rho_1=\mathtt{refresh}(\Gamma(\mathtt{x}_1))\qquad\rho_2=\mathtt{refresh}(\Gamma(\mathtt{x}_2))}{\Gamma\vdash_{\mathcal{P}}\langle(\mathtt{x}_1,\mathtt{x}_2)\mid\mathtt{typ}\rangle\Rightarrow(\rho_1,\rho_2)}$$

$$[\textsc{Proj}]\ \frac{\Sigma=\mathsf{tally}(\{\Gamma(\mathtt{x})\ \dot{\leq}\ \alpha\times\beta\})}{\Gamma\vdash_{\mathcal{P}}\langle\pi_i\mathtt{x}\mid\mathtt{typ}\rangle\Rightarrow\pi(\Sigma)}\ \begin{array}{l}\Sigma\neq\varnothing\\ \alpha,\beta\in\mathcal{V}_P\text{ fresh}\end{array}$$

$$[\textsc{App}]\ \frac{\begin{array}{c}t_1=\Gamma(\mathtt{x}_1)\qquad t_2=\Gamma(\mathtt{x}_2)\qquad\rho_1=\mathtt{refresh}(t_1)\\ \rho_2=\mathtt{refresh}(t_2)\qquad\Sigma=\mathsf{tally}(\{t_1\rho_1\ \dot{\leq}\ t_2\rho_2\to\alpha\})\end{array}}{\Gamma\vdash_{\mathcal{P}}\langle\mathtt{x}_1\mathtt{x}_2\mid\mathtt{typ}\rangle\Rightarrow @(\{\sigma\circ\rho_1\mid\sigma\in\Sigma\},\{\sigma\circ\rho_2\mid\sigma\in\Sigma\})}\ \begin{array}{l}\Sigma\neq\varnothing\\ \alpha\in\mathcal{V}_P\text{ fresh}\end{array}$$

$$[\textsc{Case}_{\mathbb{0}}]\ \frac{\sigma\in\mathsf{tally}(\{\Gamma(\mathtt{x})\ \dot{\leq}\ \mathbb{0}\})}{\Gamma\vdash_{\mathcal{P}}\langle(\mathtt{x}\in\tau)\,?\,\mathtt{x}_1:\mathtt{x}_2\mid\mathtt{typ}\rangle\Rightarrow\mathbb{0}(\{\sigma\})}$$

$$[\textsc{Case}_1]\ \frac{\sigma\in\mathsf{tally}(\{\Gamma(\mathtt{x})\ \dot{\leq}\ \tau\})}{\Gamma\vdash_{\mathcal{P}}\langle(\mathtt{x}\in\tau)\,?\,\mathtt{x}_1:\mathtt{x}_2\mid\mathtt{typ}\rangle\Rightarrow\in_1(\{\sigma\})}\ \mathtt{x}_1\in\mathrm{dom}(\Gamma)$$

$$[\textsc{Case}_2]\ \frac{\sigma\in\mathsf{tally}(\{\Gamma(\mathtt{x})\ \dot{\leq}\ \neg\tau\})}{\Gamma\vdash_{\mathcal{P}}\langle(\mathtt{x}\in\tau)\,?\,\mathtt{x}_1:\mathtt{x}_2\mid\mathtt{typ}\rangle\Rightarrow\in_2(\{\sigma\})}\ \mathtt{x}_2\in\mathrm{dom}(\Gamma)$$

$$[\textsc{Lambda}]\ \frac{\Gamma,x:\mathbf{u}\vdash_{\mathcal{P}}\langle\kappa\mid\mathcal{K}\rangle\Rightarrow\Bbbk}{\Gamma\vdash_{\mathcal{P}}\langle\lambda x.\kappa\mid\lambda(\mathbf{u},\mathcal{K})\rangle\Rightarrow\lambda(\mathbf{u},\Bbbk)}$$

$$[\textsc{Var}]\ \frac{\rho = \mathtt{refresh}(\Gamma(\mathtt{x}))}{\Gamma \vdash_{\mathcal{P}} \langle \mathtt{x} \mid \mathtt{typ} \rangle : \rho} \qquad [\textsc{BindSkip}]\ \frac{\Gamma \vdash_{\mathcal{P}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathbb{k}}{\Gamma \vdash_{\mathcal{P}} \langle \mathtt{bind}\, \mathtt{x} = a\, \mathtt{in}\, \kappa \mid \mathtt{skip}\ (\mathcal{K}) \rangle \Rightarrow \mathtt{skip}\ \mathbb{k}}\ \mathtt{x} \notin \mathrm{dom}(\Gamma)$$

$$[\textsc{BindKeep}]\ \frac{\Gamma \vdash_{\mathcal{P}} \langle a \mid \mathcal{A} \rangle \Rightarrow \mathbb{a} \qquad \Gamma \vdash_{\mathcal{A}} [a \mid \mathbb{a}] : s \qquad (\forall i \in I)\ \ \Gamma, \mathtt{x} : s \wedge \mathbf{u}_i \vdash_{\mathcal{P}} \langle \kappa \mid \mathcal{K}_i \rangle \Rightarrow \mathbb{k}_i}{\Gamma \vdash_{\mathcal{P}} \langle \mathtt{bind}\, \mathtt{x} = a\, \mathtt{in}\, \kappa \mid \mathtt{keep}\ (\mathcal{A}, \varnothing, \{(\mathbf{u}_i, \mathcal{K}_i)\}_{i \in I}) \rangle \Rightarrow \mathtt{keep}\ (\mathbb{a}, \{(\mathbf{u}_i, \mathbb{k}_i)\}_{i \in I})}\ (*)$$

where $(*)$ is $I \neq \varnothing$ and $\bigvee_{i \in I} \mathbf{u}_i \simeq \mathbb{1}$.

$$[\textsc{Inter}]\ \frac{(\forall i \in I) \quad \Gamma \vdash_{\mathcal{P}} \langle \eta \mid \mathcal{H}_i \rangle \Rightarrow \mathbb{h}_i}{\Gamma \vdash_{\mathcal{P}} \langle \eta \mid \ \bigwedge(\varnothing, \{\mathcal{H}_i\}_{i \in I}) \rangle \Rightarrow \bigwedge(\{\mathbb{h}_i\}_{i \in I})}\ I \neq \varnothing$$

The following rules can be added to support the extensions presented in Appendix A:

$$[\textsc{Let}]\ \frac{}{\Gamma \vdash_{\mathcal{P}} \langle \mathtt{let}\, \mathtt{x}_1\, \mathtt{in}\, \mathtt{x}_2 \mid \mathtt{typ} \rangle \Rightarrow \varnothing} \qquad [\textsc{Constr}]\ \frac{\Sigma = \mathsf{tally}(\{\Gamma(\mathtt{x}) \mathrel{\dot{\leq}} \tau\})}{\Gamma \vdash_{\mathcal{P}} \langle \mathtt{x} \mathbin{⨾} \tau \mid \mathtt{typ} \rangle \Rightarrow ⨾(\Sigma)}$$

H.3 Split propagation

The split propagation system defined in this section tries to deal with the following problem: *given a current environment* Γ*, an atom* a *and a type* t*, what additional assumptions can I make on* Γ *in order to ensure that* a *has type* t*?* It is used by the main reconstruction system in order to propagate splits made by bindings.

We note $\lceil t \rceil$ the instance of t obtained by applying the following substitutions:

- Any polymorphic variable in t only appearing in covariant positions is substituted by $\mathbb{1}$,
- Any polymorphic variable in t only appearing in contravariant positions is substituted by $\mathbb{0}$.

$$[\textsc{Const}_1]\ \frac{b_c \leq \mathbf{u}}{\Gamma \vdash_{\mathcal{E}} (c : \mathbf{u}) \Rightarrow \{\varnothing\}} \qquad [\textsc{Const}_2]\ \frac{}{\Gamma \vdash_{\mathcal{E}} (c : \mathbf{u}) \Rightarrow \{\}}$$

$$[\textsc{Ax}_1]\ \frac{\Gamma(x) \leq \mathbf{u}}{\Gamma \vdash_{\mathcal{E}} (x : \mathbf{u}) \Rightarrow \{\varnothing\}} \qquad [\textsc{Ax}_2]\ \frac{}{\Gamma \vdash_{\mathcal{E}} (x : \mathbf{u}) \Rightarrow \{\}}$$

$$[\textsc{Proj}_1]\ \frac{}{\Gamma \vdash_{\mathcal{E}} (\pi_1 \mathtt{x} : \mathbf{u}) \Rightarrow \{\{\mathtt{x} : \mathbf{u} \times \mathbb{1}\}\}} \qquad [\textsc{Proj}_2]\ \frac{}{\Gamma \vdash_{\mathcal{E}} (\pi_2 \mathtt{x} : \mathbf{u}) \Rightarrow \{\{\mathtt{x} : \mathbb{1} \times \mathbf{u}\}\}}$$

$$[\textsc{Pair}]\ \frac{\mathbf{u} \wedge (\mathbb{1} \times \mathbb{1}) \overset{\text{DNF}}{\simeq} \bigvee_{i \in I} (\mathbf{u}_i \times \mathbf{v}_i)}{\Gamma \vdash_{\mathcal{E}} ((\mathtt{x}_1, \mathtt{x}_2) : \mathbf{u}) \Rightarrow \{\{\mathtt{x}_1 : \mathbf{u}_i\} \wedge \{\mathtt{x}_2 : \mathbf{v}_i\} \mid i \in I\}}$$

$$[\textsc{Case}]\ \frac{}{\Gamma \vdash_{\mathcal{E}} ((\mathtt{x} \in \tau)\, ?\, \mathtt{x}_1 : \mathtt{x}_2 : \mathbf{u}) \Rightarrow \{\{\mathtt{x} : \tau,\ \mathtt{x}_1 : \mathbf{u}\}, \{\mathtt{x} : \neg\tau,\ \mathtt{x}_2 : \mathbf{u}\}\}}$$

$$[\textsc{App}]\ \frac{\{\sigma_i\}_{i \in I} = \mathsf{tally}(\{\Gamma(\mathtt{x}_1) \mathrel{\dot{\leq}} \alpha \to \mathbf{u}\}) \qquad \mathbb{\Gamma} = \{\{\mathtt{x}_2 : \lceil \alpha\sigma_i \rceil\} \mid i \in I\}}{\Gamma \vdash_{\mathcal{E}} (\mathtt{x}_1 \mathtt{x}_2 : \mathbf{u}) \Rightarrow \{\Gamma \in \mathbb{\Gamma} \mid \mathsf{vars}(\Gamma) \subseteq \mathcal{V}_M\}}\ \alpha \text{ fresh}$$

$$[\textsc{Lambda}]\ \frac{}{\Gamma \vdash_{\mathcal{E}} (\lambda x.\, \kappa : \mathbf{u}) \Rightarrow \{\}}$$

The following rules can be added to support the extensions presented in Appendix A:

$$[\textsc{Let}]\ \frac{}{\Gamma \vdash_{\mathcal{E}} (\mathtt{let}\,\mathtt{x}_1\,\mathtt{in}\,\mathtt{x}_2\ : \mathbf{u}) \Rightarrow \{\{\mathtt{x}_2 : \mathbf{u}\}\}} \qquad [\textsc{Constr}]\ \frac{}{\Gamma \vdash_{\mathcal{E}} (\mathtt{x} \mathbin{⨾} \tau : \mathbf{u}) \Rightarrow \{\{\mathtt{x} : \mathbf{u}\}\}}$$

I PROOFS

The proofs are for the source language presented in section 2 without extension:

$$\textbf{Expressions} \quad e \quad ::= \quad c \mid x \mid \lambda x.e \mid ee \mid (e,e) \mid \pi_i e \mid (e{\in}\tau)\,?\,e:e$$

We recall some notations relative to substitutions:

- ρ ranges over renamings of polymorphic variables, that is, injective substitutions from $\mathcal{V}_P$ to $\mathcal{V}_P$
- σ ranges over substitutions from polymorphic type variables $\mathcal{V}_P$ to types
- Σ ranges over sets of substitutions from polymorphic type variables $\mathcal{V}_P$ to types
- ψ ranges over substitutions from monomorphic type variables $\mathcal{V}_M$ to monomorphic types
- Ψ ranges over sets of substitutions from monomorphic type variables $\mathcal{V}_M$ to monomorphic types

I.1 Declarative type system

I.1.1 Modifications to the declarative type system. See Appendix D for the full declarative system, without the rules for extensions.

In the next subsection (I.1.2), we will define normalized forms for derivations of the declarative type system. These normalized form are not unique, but still satisfies properties that will be used to prove the safety of the declarative type system, as well as the completeness of the declarative type system towards the declarative one.

In order to be able to express these normalized forms, we first need to slightly modify the declarative type system (of course, all all the changes made are admissible). **All the proofs (in this section and the following ones) will refer to this modified version of the declarative type system.**

First, we modify the [Ax] rule so that it can perform a renaming of the polymorphic type variables of the returned type:

$$[\text{Ax}]\ \frac{}{\Gamma \vdash x : \Gamma(x)\rho}$$

This new [Ax] rule can be derived in the initial delcarative type system by composing a [Ax] rule and a [Inst] rule. However, it makes sense to allow the [Ax] rule to perform a renaming directly so that it can avoid correlation between two types without resorting to the [Inst] rule (that will only be used before destructor). For instance, we want to be able to type the pair (f, f), with f being a variable with type $\alpha \to \alpha$, with the type $(\alpha \to \alpha) \times (\beta \to \beta)$ and without having to use a [Inst] rule.

Secondly, we will use [$\vee$] and [$\wedge$] rules of multiple arity instead of the binary ones:

$$[\vee]\ \frac{\Gamma \vdash e' : s \qquad (\forall i \in I) \quad \Gamma, x : s \wedge \mathbf{u}_i \vdash e : t}{\Gamma \vdash e\{e'/x\} : t}\ \bigvee_{i\in I} \mathbf{u}_i \simeq \mathbb{1} \qquad [\wedge]\ \frac{(\forall i \in I) \quad \Gamma \vdash e : t_i}{\Gamma \vdash e : \bigwedge_{i\in I} t_i}\ I \neq \varnothing$$

The new [$\wedge$] rule can be derived in the previous system by composing sevral [$\wedge$] rules. The new [$\vee$] rule, however, is admissible in a more subtle way:

- From any decomposition $\bigvee_{i\in I} \mathbf{u}_i$ covering $\mathbb{1}$, we can construct a partition $\bigvee_{j\in J} \mathbf{u}'_j$ of $\mathbb{1}$ with $|J| \geq 2$ and such that $\forall j \in J.\ \exists i \in I.\ \mathbf{u}'_j \leq \mathbf{u}_i$.
- A decomposition into a partition $\{\mathbf{u}'_1, \mathbf{u}'_2, \mathbf{u}'_3\}$ of $\mathbb{1}$ can easily be obtained be composing two binary $[\vee]$ rules:

$$[\vee]\ \dfrac{\dfrac{\cdots}{\Gamma \vdash e' : s} \qquad \dfrac{\cdots}{\Gamma, y : s \wedge \mathbf{u}'_1 \vdash e\{y/x\} : t} \qquad \dfrac{X}{\Gamma, y : s \wedge \neg\mathbf{u}'_1 \vdash e\{y/x\} : t}}{\Gamma \vdash e\{y/x\}\{e'/y\} : t}$$

with X being the following derivation:

$$[\vee]\ \dfrac{\dfrac{[\text{Ax}]}{\Gamma, y : s \wedge \neg\mathbf{u}'_1 \vdash y : s \wedge \neg\mathbf{u}'_1} \qquad \dfrac{\cdots}{\Gamma, y : s \wedge \neg\mathbf{u}'_1, x : s \wedge \mathbf{u}'_2 \vdash e : t} \qquad \dfrac{\cdots}{\Gamma, y : s \wedge \neg\mathbf{u}'_1, x : s \wedge \mathbf{u}'_3 \vdash e : t}}{\Gamma, y : s \wedge \neg\mathbf{u}'_1 \vdash e\{y/x\} : t}$$

This construction can be generalized for any partition of $\mathbb{1}$ of cardinality at least 2.

- Every premise in the construction above is derivable from one of the $\Gamma, x : s \wedge \mathbf{u}_i \vdash e : t$ of the initial derivation by alpha-renaming and monotonicity (see I.4 below).

Lastly, we will distinguish variables that are introduced by a [→I] from variables introduced by a [∨]: we will use binding variables (x, y, z) for the latter. More formally, the syntax of expressions is extended, and the rules changed accordingly:

$$\textbf{Expressions} \quad e \quad ::= \quad c \mid x \mid \mathrm{x} \mid \lambda x.e \mid ee \mid (e,e) \mid \pi_i e \mid (e{\in}\tau)\ ?\ e : e \tag{39}$$

$$[\text{Ax}_\lambda]\ \dfrac{}{\Gamma \vdash x : \Gamma(x)\rho} \qquad [\text{Ax}_\vee]\ \dfrac{}{\Gamma \vdash \mathrm{x} : \Gamma(\mathrm{x})\rho}$$

$$[\vee]\ \dfrac{\Gamma \vdash e' : s \qquad (\forall i \in I) \quad \Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : t}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}\ \bigvee_{i\in I} \mathbf{u}_i \simeq \mathbb{1}$$

Note that this is still equivalent to the initial type system as both binding variables and lambda variables are treated the same way ($[\text{Ax}_\lambda]$ and $[\text{Ax}_\vee]$ both correspond to the [Ax] rule we defined earlier).

Also, we will call *structural rules* the rules [Const], $[\text{Ax}_\lambda]$, [→I], [→E], [×I], $[\times\text{E}_1]$, $[\times\text{E}_2]$, $[\mathbb{0}]$, $[\in_1]$ and $[\in_2]$. Note that the rule $[\text{Ax}_\vee]$ is not considered structural.

I.1.2 Normalisation Lemmas. In the proofs below, we will sometimes omit the guardians of some rules in the derivations. This is only for concision and clarity, and only trivially verified guardians will be omitted.

Lemma I.1. *Any derivation $\Gamma \vdash e : t$ can be transformed into a derivation $\Gamma \vdash e : t\rho$ (for any renaming ρ on polymorphic type variables) without changing the structure of the derivation.*

Proof. Any polymorphic type variable in t must be introduced either by an axiom or a [Inst] rule. Thus, we can derive $\Gamma \vdash e : t\rho$ with the following transformations:

- The renaming ρ' of any axiom rule is replaced by the renaming $\rho \circ \rho'$
- The substitution σ of any [Inst] rule is replaced by the substitution $\rho \circ \sigma \circ \rho^{-1}$

□

Definition I.2. *We define the order relation $\leq_P$ on types as follows (with Σ a set of substitutions from $\mathcal{V}_P$ to types):*

$$t_1 \leq_P t_2 \Leftrightarrow \exists\Sigma.\ t_1\Sigma \leq t_2$$

Definition I.3. *We define the order relation $\leq_P$ on environments as follows:*

$$\Gamma_1 \leq_P \Gamma_2 \Leftrightarrow \forall x \in dom(\Gamma_2).\ x \in dom(\Gamma_1) \text{ and } \Gamma_1(x) \leq_P \Gamma_2(x)$$

where x denotes both lambda variables and binding variables (usually denoted by x).

Lemma I.4 (Monotonicity). *If $\Gamma \vdash e : t$ and $\Gamma' \leq_P \Gamma$, then $\Gamma' \vdash e : t$.*

Proof. We apply the rules [Inst], [$\wedge$] and [$\leq$] just after the axiom rules of the original derivation whenever required. □

Lemma I.5 (Generation of an arbitrary [$\vee$] rule). *Let Γ a type environment, e and e_{x} two expressions, and $\{\boldsymbol{u}_i\}_{i\in I}$ a set of monomorphic types such that $\bigvee_{i\in I} \boldsymbol{u}_i \simeq \mathbb{1}$. Let D be a derivation for the judgement $\Gamma \vdash e\{e_{\mathsf{x}}/\mathsf{x}\} : t$ such that D does not contain any [$\vee$] rule that performs a substitution $\{e_{\mathsf{y}}/\mathsf{y}\}$ with e_{y} being a strict subexpression of e_{x}. If e_{x} is typable under the context Γ, then D can be transformed so that it ends with an application of the [$\vee$] rule of the following form:*

$$[\vee]\ \dfrac{\dfrac{\cdots}{\Gamma \vdash e_{\mathsf{x}} : s} \qquad \dfrac{\cdots}{\Gamma, \mathsf{x} : s \wedge \boldsymbol{u}_i \vdash e : t}\ \forall i \in I}{\Gamma \vdash e\{e_{\mathsf{x}}/\mathsf{x}\} : t}$$

for some type s that cannot be choosen arbitrarily.

This transformation does not add any new structural rule nor [$\vee$] rule (except the one at the root) to the derivation.

Proof. Let's call C a derivation for $\Gamma \vdash e_{\mathsf{x}} : \mathbb{1}$. First, we collect in D the set $\{C_k\}_{k\in K}$ of all the subderivations for e_{x}. By noticing that no variable in $\mathsf{fv}(e_{\mathsf{x}})$ could have been introduced by a lambda in e (we recall that all our substitutions are capture-avoiding), we know that these derivations are valid under the environment Γ.

Then, we build the following derivation:

$$[\vee]\ \dfrac{[\wedge]\ \dfrac{\dfrac{C}{\Gamma \vdash e_{\mathsf{x}} : \mathbb{1}} \qquad \dfrac{C_k}{\Gamma \vdash e_{\mathsf{x}} : t_k}\ \forall k \in K}{\Gamma \vdash e_{\mathsf{x}} : \bigwedge_{k\in K} t_k} \qquad \dfrac{D'_i}{\Gamma, \mathsf{x} : (\bigwedge_{k\in K} t_k) \wedge \mathbf{u}_i \vdash e : t}\ \forall i \in I}{\Gamma \vdash e\{e_{\mathsf{x}}/\mathsf{x}\} : t}$$

with each D'_i being a derivation easily derived from D by substituting e_{x} by x when relevant, using an axiom rule on x instead of a subderivation for e_{x} when necessary, and by using monotonicity (I.4). The hypothesis on the derivation D ensures that it does not contain any conflicting [$\vee$] rule that would become inapplicable due to the fact that e_{x} has been substituted by x. □

Lemma I.6 (Deletion of aliasing). *In any derivation, any occurrence of a [$\vee$] rule applying a substitution $\{\mathsf{x}/\mathsf{y}\}$ can be removed (without adding any other [$\vee$] rule nor structural rule in the derivation).*

Proof. The following transformation can be performed:

$$[\vee]\ \frac{\dfrac{A}{\Gamma \vdash e' : s} \qquad \dfrac{\dfrac{[\vee]\ \dfrac{\dfrac{B}{\Gamma'' \vdash \mathsf{x} : s''} \quad \dfrac{C_j \quad \forall j \in J}{\Gamma'', \mathsf{y} : s'' \wedge \mathbf{u}''_j \vdash e'' : t''}}{\Gamma'' \vdash e''\{\mathsf{x}/\mathsf{y}\} : t''}}{\cdots}}{\Gamma, \mathsf{x} : s \wedge \mathbf{u}_k \vdash e : t} \qquad \dfrac{D_i}{\Gamma, \mathsf{x} : s \wedge \mathbf{u}_i \vdash e : t}\ \forall i \in I \setminus \{k\}}{\Gamma \vdash e\{e'/\mathsf{x}\} : t}$$

$$\text{with } \Gamma'' = (\Gamma, \mathsf{x} : s \wedge \mathbf{u}_k) \uplus \Gamma'$$

$$\downarrow$$

$$[\vee]\ \frac{\dfrac{A}{\Gamma \vdash e' : s} \qquad \dfrac{\dfrac{\dfrac{C'_j}{(\Gamma, \mathsf{x} : s \wedge \mathbf{u}_k \wedge \mathbf{u}''_j) \uplus \Gamma' \vdash e''\{\mathsf{x}/\mathsf{y}\} : t''}}{\cdots}}{\Gamma, \mathsf{x} : s \wedge \mathbf{u}_k \wedge \mathbf{u}''_j \leq s'' \wedge \mathbf{u}''_j \vdash e : t}\ \forall j \in J \qquad \dfrac{D_i}{\Gamma, \mathsf{x} : s \wedge \mathbf{u}_i \vdash e : t}\ \forall i \in I \setminus \{k\}}{\Gamma \vdash e\{e'/\mathsf{x}\} : t}$$

where C'_j is easily derived from C_j by replacing occurrences of $[\text{Ax}_\vee]$ on y by $[\text{Ax}_\vee]$ on x and applying the monotonicity lemma (I.4). Note that we have $s \wedge \mathbf{u}_k \leq s''$ as the derivation B can only derive for x a type larger than $\Gamma''(\mathsf{x})$, and thus $s \wedge \mathbf{u}_k \wedge \mathbf{u}''_j \leq s'' \wedge \mathbf{u}''_j$. $\square$

For the following lemmas, we fix an arbitrary total order $\leq$ over the expressions of the source language modulo alpha-renaming. This order must be an extension of the subexpression order (i.e. if e_1 is a subexpression of e_2 modulo alpha-renaming, then we should have $e_1 \leq e_2$).

Then, we introduce for convenience the notation $[\vee\text{Pat}]$ for denoting, given a derivation D, a segment of a branch of D that is only composed of a succession of $[\vee]$ rules, the one connected to the next by any of its premises except the first one, and that is compatible with the $\leq$ order in the following way: if a rule of this segment is doing the substitution $\{e_\mathsf{x}/\mathsf{x}\}$ and if its premise is also in the segment and is doing the substitution $\{e_\mathsf{y}/\mathsf{y}\}$, then $\lceil e_\mathsf{x} \rceil \leq \lceil e_\mathsf{y} \rceil$, where $\lceil e \rceil$ is abusively used here to denote the expression obtained by applying to e all the substitutions made by the $[\vee]$ rules crossed when going toward the root of D.

We also introduce two new notations that take the form of two new rules, but are actually just shortands for a specific pattern:

$$[\text{Inst}\wedge\leq]\ \frac{\dfrac{A}{\Gamma \vdash e : t'} \qquad (\exists \Sigma.\ \textstyle\bigwedge_{\sigma \in \Sigma} t'\sigma \leq t)}{\Gamma \vdash e : t} \quad \leftrightarrow \quad [\leq]\ \frac{[\wedge]\ \dfrac{[\text{Inst}]\ \dfrac{\dfrac{A}{\Gamma \vdash e : t'}}{\Gamma \vdash e : t'\sigma}\ \forall \sigma \in \Sigma}{\Gamma \vdash e : \bigwedge_{\sigma \in \Sigma} t'\sigma}}{\Gamma \vdash e : t}$$

$$[\to\text{I}\wedge]\ \frac{\dfrac{A_i}{\Gamma, x : \mathbf{u}_i \vdash e : t_i}\ \forall i \in I}{\Gamma \vdash \lambda x.e : \bigwedge_{i \in I} \mathbf{u}_i \to t_i} \quad \leftrightarrow \quad [\wedge]\ \frac{[\to\text{I}]\ \dfrac{\dfrac{A_i}{\Gamma, x : \mathbf{u}_i \vdash e : t_i}}{\Gamma \vdash \lambda x.e : \mathbf{u}_i \to t_i}\ \forall i \in I}{\Gamma \vdash \lambda x.e : \bigwedge_{i \in I} \mathbf{u}_i \to t_i}$$

The normalisation lemmas that follow are cumulative: each one can be combined with the next ones (the transformations introduced by a lemma do not break the properties of the previous lemmas).

Lemma I.7 (Normalisation of [$\vee$]). *Any derivation of $\Gamma \vdash e : t$ can be transformed so that:*

- *every occurrence of a structural rule except [$\rightarrow$I] does not have any structural rule (including [$\rightarrow$I]) in the subderivations of its premises, and*
- *the premise of any occurrence of a [$\rightarrow$I] rule is either a [$\vee$] rule or its subderivation does not contain any structural rule, and*
- *any premise except the first of any occurrence of a [$\vee$] rule is either another [$\vee$] rule or its subderivation does not contain any structural rule*

and such that every occurrence of [$\vee$] rule doing the substitution $e'\{e_{\mathsf{x}}/\mathsf{x}\}$ is such that:

- *e_{x} is not a subexpression of e' (maximal sharing), and*
- *e_{x} is not a binding variable (no aliasing), and*
- *x is a subexpression of e' (no useless binding)*

and is part of a [$\vee$Pat] pattern that starts either:

- *At the root of the derivation (optionally preceded by a [Inst$\wedge\leq$] pattern), or*
- *At the premise of a [$\rightarrow$I] rule that introduces a variable y such that $y \in \mathsf{fv}(e_{\mathsf{x}})$, or*
- *At the n-th premise ($n \geq 2$) of a [$\vee$] rule that introduces a variable y such that $\mathsf{y} \in \mathsf{fv}(e_{\mathsf{x}})$.*

Proof. We can transform any derivation into a derivation that satisfies these properties.
First, we can remove any aliasing by applying I.6 as needed. We can also trivially remove useless bindings (i.e. those doing a substitution $e'\{e_{\mathsf{x}}/\mathsf{x}\}$ where e' does not contain x).
Then, let's consider, in the whole derivation, all the rules that satisfy one of these:

- It is a structural rule that is in the subderivation of a premise of another structural rule except [$\rightarrow$I],
- It is a structural rule in the subderivation of the premise of a [$\rightarrow$I] rule with no [$\vee$] rule at the root,
- It is a structural rule in the subderivation of the n-th premise ($n \geq 2$) of a [$\vee$] rule with no other [$\vee$] rule at the root,
- It is a [$\vee$] rule such that e_{x} is a subexpression of e',
- It is a [$\vee$] rule that is not part of a [$\vee$Pat] pattern as described in the statement.

If there is no such rule, then we are done (the properties of this lemma are satisfied). Otherwise, to each of these faulty rules, we associate an expression:

- For a structural rule applied on an expression e', we associate e'
- For a [$\vee$] rule doing the substitution $\{e_{\mathsf{x}}/\mathsf{x}\}$, we associate e_{x}

Now, we select among those rules the one that minimizes (according to the order $\leq$) $\lceil e' \rceil$, with e' its associated expression and $\lceil e \rceil$ denoting the expression e to which we apply all the substitutions made by the [$\vee$] rules crossed when going toward the root of the derivation. Let's call this rule R and the associated expression e'.
Now, let's locate, in the segment from the root to R, the rule R' nearest from the root such that:

- All lambda variables and binding variables in e' are defined in the environment of the judgement of R',
- R' is not a [$\vee$] rule, or it makes a substitution $\{e_{\mathsf{y}}/\mathsf{y}\}$ with $\lceil e' \rceil \leq \lceil e_{\mathsf{y}} \rceil$

Let's note $\Gamma_{R'} \vdash e_{R'} : t_{R'}$ the judgement of R', and let's consider the associated subderivation. Note that, in this subderivation, there is no [$\vee$] rule that makes a substitution $\{e_{\mathsf{y}}/\mathsf{y}\}$ with e_{y} a strict subexpression of e', because $\lceil e_{\mathrm{y}} \rceil$ would be smaller than $\lceil e' \rceil$, thus breaking the properties of the lemma and contradicting the minimality of $\lceil e' \rceil$. Also note that this subderivation contains R.

We apply the lemma I.5 on the root of this subderivation (R') so that it performs the substitution $\bar{e_{R'}}\{e'/\mathrm{z}\}$, with z fresh and $\bar{e_{R'}} = e_{R'}\{\mathrm{z}/e'\}$, and using the decomposition $\{\mathbf{u}_i\}_{i \in I} = \{\mathbb{1}\}$. This lemma can be applied as:

- There cannot be in our subderivation any $[\vee]$ rule substituting a strict subexpression of e',
- We know that $\Gamma_{R'} \vdash e' : \mathbb{1}$ holds: we can derive it from the first premise of R if R is a $[\vee]$ rule, or from R itself if R is a structural rule

Now, in this new subderivation, if R is a $[\vee]$ rule, then it can be removed by using lemma I.6 (as well as other aliasing that would have been introduced in other branches). Not that, if R is a structural rule, it has already been eliminated by the application of I.5.

Finally, we put our subderivation back in its place in the original derivation. Note that this process does not introduce any new rule that would break the properties of the lemma. However, an already existing $[\vee]$ rule could start breaking the lemma due to this process if it was part of a [∨Pat] pattern starting on R, but in this case we know that its assicated expression e'' is such that $\lceil e' \rceil \leq \lceil e'' \rceil$. Thus, we can conclude by repeating this process until all the rules satisfy the properties of the lemma. □

Lemma I.8 (Normalisation of [Inst]). *Any derivation of $\Gamma \vdash e : t$ can be transformed so that every application of [Inst] is part of a [Inst∧≤] pattern that is either:*

- *At the root of the derivation, or*
- *The first premise of a $[\mathbb{0}]$, $[\in_1]$ or $[\in_2]$ rule, or*
- *The premise of a $[\times E_1]$ or $[\times E_2]$ rule, or*
- *One of the premises of a $[\to E]$ rule*

Proof. First, we apply the normalisation lemma I.7 on the derivation.

Now, we proceed by induction on $(n_\vee, n)$ (using the lexicographic order), with $n_\vee$ the number of $[\vee]$ rules in the derivation, and n the total number of rules in the derivation.

The base case is trivial.

For the inductive case, we consider the root of the derivation.

If the root is a [Inst] or $[\leq]$, we apply the induction hypothesis to its premise:

- If the new premise does not end with a [Inst∧≤] pattern, then we are already done (note that a single [Inst] or $[\leq]$ is a particular cases of a [Inst∧≤] pattern).
- If the new premise ends with a [Inst∧≤] pattern, then we can trivially merge them together into a single [Inst∧≤] pattern.

If the root is a $[\wedge]$, we apply the induction hypothesis to all its premises:

- If none of its new premises end with a [Inst∧≤] pattern, then we are already done.
- If (at least) one of its new premises ends with a [Inst∧≤] pattern, we can consider without loss of generality that all its premises end with a [Inst∧≤] pattern. We then apply the following transformation (with $\Sigma = \bigcup_{i \in I} \Sigma_i$):

$$[\wedge]\ \dfrac{[\textsc{Inst}\wedge\leq]\ \dfrac{\dfrac{A_i}{\Gamma \vdash e : t'_i} \qquad (\bigwedge_{\sigma\in\Sigma_i} t'_i\sigma \leq t_i)}{\Gamma \vdash e : t_i}\ \forall i \in I}{\Gamma \vdash e : \bigwedge_{i\in I} t_i}$$

$$\downarrow$$

$$[\textsc{Inst}\wedge\leq]\ \dfrac{[\wedge]\ \dfrac{\dfrac{A_i}{\Gamma \vdash e : t'_i}\ \forall i \in I}{\Gamma \vdash e : \bigwedge_{i\in I} t'_i} \qquad \left(\begin{array}{l}\bigwedge_{\sigma\in\Sigma}(\bigwedge_{i\in I} t'_i)\sigma \ \leq \bigwedge_{i\in I}\bigwedge_{\sigma\in\Sigma_i}(\bigwedge_{i\in I} t'_i)\sigma \\ \leq \bigwedge_{i\in I}\bigwedge_{\sigma\in\Sigma_i} t'_i\sigma \leq \bigwedge_{i\in I} t_i\end{array}\right)}{\Gamma \vdash e : \bigwedge_{i\in I} t_i}$$

If the root is a $[\vee]$:

- We first apply the induction hypothesis to its first premise. If its new first premise ends with a $[\textsc{Inst}\wedge\leq]$ pattern, we apply the following transformation (otherwise, we continue to the next step):

$$[\vee]\ \dfrac{[\textsc{Inst}\wedge\leq]\ \dfrac{\dfrac{A}{\Gamma \vdash e' : s'} \qquad (\bigwedge_{\sigma\in\Sigma} s'\sigma \leq s)}{\Gamma \vdash e' : s} \qquad \dfrac{B_i}{(\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i) \vdash e : t}\ \forall i \in I}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}$$

$$\downarrow$$

$$[\vee]\ \dfrac{\dfrac{A}{\Gamma \vdash e' : s'} \qquad \dfrac{B'_i}{(\Gamma, \mathrm{x} : s' \wedge \mathbf{u}_i \leq_{\mathrm{P}} s \wedge \mathbf{u}_i) \vdash e : t}\ \forall i \in I}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}$$

  with B'_i a derivation easily derived from B_i by monotonicity (I.4). Note that the application of the monotonicity lemma might add unwanted occurrences of a $[\textsc{Inst}\wedge\leq]$ pattern, which will be eliminated with the next step.
- The next step is to apply the induction hypothesis on the other (new) premises of this $[\vee]$ rule. If at least one of these new premises is ending with a $[\textsc{Inst}\wedge\leq]$ pattern, we can consider without loss of generality that it is the case for all of them. We can also suppose that the types in the conclusion of these premises all have disjoint polymorphic type variables: if it is not the case, it can be ensured by applying I.1 to these premises and adding an instantiation at the root to compensate. Then, we apply the following transformation:

$$[\vee]\ \dfrac{\dfrac{A}{\Gamma \vdash e' : s} \qquad [\textsc{Inst}\wedge\leq]\ \dfrac{\dfrac{B_i}{\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : t_i} \quad (\bigwedge_{\sigma\in\Sigma_i} t_i\sigma \leq t)}{\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : t}\ \forall i \in I}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}$$

$$\downarrow$$

$$[\textsc{Inst}\wedge\leq]\ \dfrac{[\vee]\ \dfrac{\dfrac{A}{\Gamma \vdash e' : s} \qquad [\leq]\ \dfrac{\dfrac{B_i}{\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : t_i}}{\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : \bigvee_{i\in I} t_i}\ \forall i \in I}{\Gamma \vdash e\{e'/\mathrm{x}\} : \bigvee_{i\in I} t_i}\quad (\star)}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}$$

($\star$) We justify this [Inst$\wedge\leq$] application by using $\Sigma = \{\sigma_1 \cup \cdots \cup \sigma_n \mid \sigma_1 \in \Sigma_1, \ldots, \sigma_n \in \Sigma_n\}$ with $I = \{1, \ldots, n\}$, and where $\cup$ designates the composition of disjoint substitutions (their disjointness can be guaranteed by the fact that we assumed every t_i to have disjoint polymorphic vars):

$$\begin{aligned}
&\textstyle\bigwedge_{\sigma\in\Sigma}(\bigvee_{i\in 1..n} t_i)\sigma \\
\simeq\ &\textstyle\bigwedge_{(\sigma_1,\ldots,\sigma_n)\in\Sigma_1\times\cdots\times\Sigma_n}(\bigvee_{i\in 1..n} t_i)(\sigma_1 \cup \cdots \cup \sigma_n) \\
\simeq\ &\textstyle\bigwedge_{(\sigma_1,\ldots,\sigma_n)\in\Sigma_1\times\cdots\times\Sigma_n} \bigvee_{i\in 1..n} t_i(\sigma_1 \cup \cdots \cup \sigma_n) \\
\simeq\ &\textstyle\bigwedge_{(\sigma_1,\ldots,\sigma_n)\in\Sigma_1\times\cdots\times\Sigma_n} \bigvee_{i\in 1..n} t_i\sigma_i \\
\simeq\ &\textstyle\bigvee_{i\in 1..n} \bigwedge_{\sigma_i\in\Sigma_i} t_i\sigma_i && \text{(distributivity of $\vee$ over $\wedge$)} \\
\leq\ &\textstyle\bigvee_{i\in 1..n} t \quad \leq t
\end{aligned}$$

The new [$\leq$] rule that appears as premise of the [$\vee$] rule could interrupt a [$\vee$Pat] pattern, thus breaking the normalisation lemma I.7. In this case, we move this [$\leq$] rule up as follows:

$$[\leq]\ \dfrac{[\vee]\ \dfrac{\dfrac{A}{\Gamma \vdash e' : s} \qquad \dfrac{B_i}{\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : t'}\ \forall i \in I}{\Gamma \vdash e\{e'/\mathrm{x}\} : t'}}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}$$

$$\downarrow$$

$$[\vee]\ \dfrac{\dfrac{A}{\Gamma \vdash e' : s} \qquad [\leq]\ \dfrac{\dfrac{B_i}{\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : t'}}{\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : t}\ \forall i \in I}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}$$

The other cases are similar or straightforward. □

Lemma I.9 (Normalisation of [$\wedge$]). *Any derivation of $\Gamma \vdash e : t$ can be transformed so that every application of [$\wedge$]:*

- *Is part of a [$\to$I$\wedge$] pattern, or*
- *Is part of a [Inst$\wedge\leq$] pattern*

Proof. First, we apply the normalisation lemmas I.7 and I.8. In particular, this ensures that our derivation does not contain any $[\wedge]$ rule that has a $[\vee]$ rule as a premise, and that if a $[\wedge]$ rule has a [Inst] rule as a premise, then it is part of a $[\text{Inst}\wedge\leq]$ pattern.

Now, we proceed by induction on the size of the derivation.

The base case is trivial.

For the inductive case, we consider the root of the derivation.

If the root is not a $[\wedge]$, we can directly conclude by induction on its premises. Thus, let's assume that the root is a $[\wedge]$.

If one if its premises is a $[\leq]$ rule, we move it towards the root by applying the following transformation, and we conclude by induction:

$$[\wedge]\ \dfrac{[\leq]\ \dfrac{\dfrac{A}{\Gamma \vdash e : t'}}{\Gamma \vdash e : t} \qquad \dfrac{B_i}{\Gamma \vdash e : t_i}\ \forall i \in I}{\Gamma \vdash e : t \wedge \bigwedge_{i\in I} t_i} \quad \to \quad [\leq]\ \dfrac{[\wedge]\ \dfrac{\dfrac{A}{\Gamma \vdash e : t'} \qquad \dfrac{B_i}{\Gamma \vdash e : t_i}\ \forall i \in I}{\Gamma \vdash e : t' \wedge \bigwedge_{i\in I} t_i}}{\Gamma \vdash e : t \wedge \bigwedge_{i\in I} t_i}$$

If one of its premises is another $[\wedge]$ rule, we can easily merge them together and conclude by induction.

If one of its premises is a $[\rightarrow\text{E}]$ rule, we know that it will also be the case for the other premises (previous normalisation lemmas prevent the cases $[\vee]$ and [Inst], and the cases for $[\wedge]$ and $[\leq]$ have been treated previously). Thus, we can apply the following transformation and conclude by induction:

$$[\wedge]\ \dfrac{[\rightarrow\text{E}]\ \dfrac{\dfrac{A_i}{\Gamma \vdash e_1 : s_i \rightarrow t_i} \qquad \dfrac{B_i}{\Gamma \vdash e_2 : s_i}}{\Gamma \vdash e_1 e_2 : t_i}\ \forall i \in I}{\Gamma \vdash e_1 e_2 : \bigwedge_{i\in I} t_i}$$

$$\downarrow$$

$$[\rightarrow\text{E}]\ \dfrac{[\leq]\ \dfrac{[\wedge]\ \dfrac{\dfrac{A_i}{\Gamma \vdash e_1 : s_i \rightarrow t_i}\ \forall i \in I}{\Gamma \vdash e_1 e_2 : \bigwedge_{i\in I}(s_i \rightarrow t_i)}}{\Gamma \vdash e_1 e_2 : (\bigwedge_{i\in I} s_i) \rightarrow (\bigwedge_{i\in I} t_i)} \qquad [\wedge]\ \dfrac{\dfrac{B_i}{\Gamma \vdash e_2 : s_i}\ \forall i \in I}{\Gamma \vdash e_2 : \bigwedge_{i\in I} s_i}}{\Gamma \vdash e_1 e_2 : \bigwedge_{i\in I} t_i}$$

The other cases are similar or straightforward. □

Lemma I.10 (Normalisation of $[\leq]$). *Any derivation of $\Gamma \vdash e : t$ can be transformed so that every application of $[\leq]$ is either:*

- *At the root of the derivation, or*
- *The first premise of a $[\in_1]$ or $[\in_2]$ rule, or*
- *The nth premise ($n \geq 2$) of a $[\vee]$ rule, or*
- *The premise of a $[\times E_1]$ or $[\times E_2]$ rule, or*
- *The first premise of a $[\rightarrow\text{E}]$ rule*

Proof. First, we apply the normalisation lemmas I.7, I.8 and I.9.

Now, we proceed by induction on $(n_\vee, n)$ (using the lexicographic order), with $n_\vee$ the number of $[\vee]$ rules in the derivation, and n the total number of rules in the derivation.

The base case is trivial.

For the inductive case, we consider the root of the derivation.

If the root is a $[\leq]$, we apply the induction hypothesis on its premise. If this new premise ends with a $[\leq]$, then we can trivially merge these two consecutive $[\leq]$ rules into one $[\leq]$ rule.

If the root is a $[\to\mathrm{I}\wedge]$ pattern, we apply the induction hypothesis to the premises of this $[\to\mathrm{I}\wedge]$ pattern. If one of these new premises end with a $[\leq]$ rule, we can assume without loss of generality that all of them do and we apply the following transformation:

$$[\to\mathrm{I}\wedge]\ \dfrac{\dfrac{[\leq]\ \dfrac{A_i}{\Gamma, x : \mathbf{u}_i \vdash e : t_i'} \quad t_i' \leq t_i}{\Gamma, x : \mathbf{u}_i \vdash e : t_i}\ \forall i \in I}{\Gamma \vdash \lambda x.e : \bigwedge_{i\in I} \mathbf{u}_i \to t_i}$$

$$\downarrow$$

$$[\leq]\ \dfrac{[\to\mathrm{I}\wedge]\ \dfrac{\dfrac{A_i}{\Gamma, x : \mathbf{u}_i \vdash e : t_i'}\ \forall i \in I}{\Gamma \vdash \lambda x.e : \bigwedge_{i\in I} \mathbf{u}_i \to t_i'}}{\Gamma \vdash \lambda x.e : \bigwedge_{i\in I} \mathbf{u}_i \to t_i} \quad \bigwedge_{i\in I} \mathbf{u}_i \to t_i' \leq \bigwedge_{i\in I} \mathbf{u}_i \to t_i$$

If the root is a $[\vee]$:

- We first apply the induction hypothesis on its first premise. If the new first premise ends with a $[\leq]$, we apply the following transformation:

$$[\vee]\ \dfrac{[\leq]\ \dfrac{\dfrac{A}{\Gamma \vdash e' : s'}}{\Gamma \vdash e' : s}\ s' \leq s \qquad \dfrac{B_i}{\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e : t}\ \forall i \in I}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}$$

$$\downarrow$$

$$[\vee]\ \dfrac{\dfrac{A}{\Gamma \vdash e' : s'} \qquad \dfrac{B_i'}{\Gamma, \mathrm{x} : s' \wedge \mathbf{u}_i \leq s \wedge \mathbf{u}_i \vdash e : t}\ \forall i \in I}{\Gamma \vdash e\{e'/\mathrm{x}\} : t}$$

  with B_i' a derivation easily derived from B_i by monotonicity (I.4). If the application of the monotonicity lemma breaks the normalisation lemma I.8 or I.9, we can apply I.8 and I.9 again on B_i'. Note that this might add unwanted occurrences of a $[\leq]$ rule, which will be eliminated with the next point.
- Then, we apply the induction hypothesis on the other (new) premises. If one of those new premises end with a $[\leq]$ rule that breaks a $[\vee\mathrm{Pat}]$ pattern (and thus the normalisation lemma I.7), we can move it up as done in the proof of I.8.

If the root is a [Inst∧≤] pattern, we apply the induction hypothesis on its premise. If this new premise ends with a [≤], we merge it with the root using the following transformation:

$$
[\textsc{Inst}\wedge\leq]\ \dfrac{[\leq]\ \dfrac{\dfrac{A}{\Gamma \vdash e : t''} \quad t'' \leq t'}{\Gamma \vdash e : t'} \qquad (\bigwedge_{\sigma\in\Sigma} t'\sigma \leq t)}{\Gamma \vdash e : t}
\quad\longrightarrow\quad
[\textsc{Inst}\wedge\leq]\ \dfrac{\dfrac{A}{\Gamma \vdash e : t''} \qquad (\bigwedge_{\sigma\in\Sigma} t''\sigma \leq t)}{\Gamma \vdash e : t}
$$

The other cases are similar or straightforward. □

Lemma I.11 (Normalisation). *Any derivation can be transformed into a derivation satisfying the properties of all the normalisation lemmas above (I.7, I.8, I.9, I.10).*

Proof. The normalisation lemmas above can be applied the one after the other in order to cumulate their properties: the transformations introduced by a one of them preserve the properties of the ones above. □

I.1.3 Parallel Semantics. One technical difficulty in the proof of the subject reduction property is that reducing an expression e might break the use of a [∨] rule. Indeed, if in the original typing derivation a rule [∨] substitutes multiple occurrences of the expression e by a variable x, reducing one occurrence of e but not the others would alter the application of this rule (the correlation between the reduced e and the other occurrences of e will be lost).

To circumvent this issue, we introduce a notion of parallel reduction which forces to reduce all occurrences of a sub-expression at the same time.

The idea is to first define reduction rules that only apply at top-level, and then define a context rule (rule [κ] below) that allows reducing under an evaluation context, but that will apply this reduction everywhere in the term.

A step of reduction for the top-level semantics is noted $\rightsquigarrow_{\top}$, and a step of reduction for the parallel semantics is noted $\rightsquigarrow_{\mathcal{P}}$.

Top-level reductions:

$$
\begin{array}{rcll}
(\lambda x.e)v & \rightsquigarrow_{\top} & e\{v/x\} & \qquad (40)\\
\pi_1(v_1, v_2) & \rightsquigarrow_{\top} & v_1 & \qquad (41)\\
\pi_2(v_1, v_2) & \rightsquigarrow_{\top} & v_2 & \qquad (42)\\
(v{\in}\tau)\ ?\ e_1 : e_2 & \rightsquigarrow_{\top} & e_1 \qquad \text{if } v \in \tau & \qquad (43)\\
(v{\in}\tau)\ ?\ e_1 : e_2 & \rightsquigarrow_{\top} & e_2 \qquad \text{if } v \in \neg\tau & \qquad (44)
\end{array}
$$

Parallel reductions:

$$
[\kappa]\ \dfrac{e \rightsquigarrow_{\top} e'}{E[e] \rightsquigarrow_{\mathcal{P}} (E[e])\{e'/e\}}
$$

Evaluation Context $E ::= [\,] \mid vE \mid Ee \mid (v, E) \mid (E, e) \mid (E{\in}\tau)\ ?\ e : e$

Here is an example of a reduction step using the parallel semantics:

$$
[\kappa]\ \dfrac{(\lambda x.\, x)\ 42 \rightsquigarrow_{\top} 42}{\texttt{if } (\lambda x.\, x)\ 42 \in \texttt{Int then } (\lambda x.\, x)\ 42 \texttt{ else } 0 \rightsquigarrow_{\mathcal{P}} \texttt{if } 42 \in \texttt{Int then } 42 \texttt{ else } 0}
$$

Notice that the rule $[\kappa]$ applies a substitution from an expression to an expression. This is formally defined as follows:

Definition I.12 (Expression substitutions). *Expression substitutions, ranged over by ρ, map an expression into another expression. The application of an expressions substitution ρ to an expression e, noted $e\rho$ is the capture avoiding replacement defined as follows:*

- *If $e' \equiv_\alpha e''$, then $e''\{e/e'\} = e$.*
- *If $e' \not\equiv_\alpha e''$, then $e''\{e/e'\}$ is inductively defined as*

$$\begin{aligned}
c\{e/e'\} &= c \\
x\{e/e'\} &= x \\
(e_1 e_2)\{e/e'\} &= (e_1\{e/e'\})(e_2\{e/e'\}) \\
(\lambda x.e_\circ)\{e/e'\} &= \lambda x.e_\circ && x \in \mathsf{fv}(e') \\
(\lambda x.e_\circ)\{e/e'\} &= \lambda x.(e_\circ\{e/e'\}) && x \notin \mathsf{fv}(e) \cup \mathsf{fv}(e') \\
(\lambda x.e_\circ)\{e/e'\} &= \lambda y.((e_\circ\{y/x\})\{e/e'\}) && x \notin \mathsf{fv}(e), x \in \mathsf{fv}(e'), y \textit{ fresh} \\
(\pi_i e_\circ)\{e/e'\} &= \pi_i(e_\circ\{e/e'\}) \\
(e_1, e_2)\{e/e'\} &= (e_1\{e/e'\}, e_2\{e/e'\}) \\
((e_1 \in t)\ ?\ e_2 : e_3)\{e/e'\} &= (e_1\{e/e'\} \in t)\ ?\ e_2\{e/e'\} : e_3\{e/e'\}
\end{aligned}$$

Notice that the expression substitutions are up to alpha-renaming and perform only one pass.

I.1.4 Subject reduction.

Property I.13. *If $\Gamma \vdash v : \tau$, then $v \in \tau$.*

Proof. Straightforward, by induction on the derivation of the judgement $\Gamma \vdash v : \tau$. Note that the case of lambda-abstractions is trivial as τ can only be $\mathbb{0} \to \mathbb{1}$. □

Lemma I.14 (Atomicity of value types). *For any derivation $\Gamma \vdash v : s$ that does not use any $[\vee]$ rule nor $[\leq]$ rule except in the subderivation of a $[\to$I$]$ rule, we can deduce that s cannot be decomposed into a non-trivial union (i.e. $s \leq \bigvee_{i \in I} s_i \implies \exists i \in I.\ s \leq s_i$).*

Proof. As v is a value, we know that the derivation does not use any destructor nor axiom rule (except maybe in the subderivation of a $[\to$I$]$ rule). It does not use any $[\vee]$ rule nor $[\leq]$ rule neither (hypothesis). In particular, this implies that s cannot contain any type variable, except under an arrow. More precisely, we can deduce that s can be constructed with the following syntax:

$$\textbf{Value Type} \quad \bar{t} \quad ::= \quad b \mid t \to t \mid \bar{t} \times \bar{t} \mid \bar{t} \wedge \bar{t}$$

By starting from the fact that a type $t_1 \to t_2$ cannot be decomposed into a non-trivial union (nor can a base type b be), we can deduce that a type constructed with the syntax above cannot be decomposed into a non-trivial union neither. □

Theorem I.15 (Subject reduction). *If $\Gamma \vdash e : t$ and $e_\circ \rightsquigarrow_\top e'_\circ$, then $\Gamma \vdash e\{e'_\circ/e_\circ\} : t$.*

Proof. We apply the normalisation lemma (I.11) to the derivation of the judgement $\Gamma \vdash e : t$, and we proceed by induction on $(d_\lambda, n_{\vee_\top}, n)$ (using the lexicographic order), with d_λ the maximum number of imbricated lambdas in e, $n_{\vee_\top}$ the number of top-level $[\vee]$ rules in the derivation (i.e. that are not in the subderivation of a $[\to$I$]$ rule), and n the total number of rules in the derivation.

Note that, even if the derivation is initially normalized, some binding variables will be substituted by values in some inductive calls, thus the properties of I.7 about structural rules will not hold. The other properties of normalisation, however, will be preserved.

We denote by ρ the substitution $\{e'_\circ/e_\circ\}$ and by e' the expression $e\rho$. If e contains no occurrence of $e_\circ$ (modulo alpha-renaming), this theorem is trivial. Thus, we will suppose in the following that e contains at least one occurrence of $e_\circ$.

We consider the root of the derivation:

[Const] Impossible case (e cannot contain any reducible expression).
[Ax$_\lambda$] Impossible case (e cannot contain any reducible expression).
[Ax$_\vee$] Impossible case (e cannot contain any reducible expression).
[$\leq$] By induction on the premise $\Gamma \vdash e : t'$ (with $t' \leq t$), we get a derivation for $\Gamma \vdash e' : t'$, thus we can derive $\Gamma \vdash e' : t$ by using [$\leq$].
[Inst] Similar to the previous case (by induction on the premise).
[$\wedge$] Similar to the previous case (by induction on the premises).
[$\to$I] We have $e' \equiv \lambda x.\ (e_1\rho)$. We can derive $\Gamma, x : \mathbf{u}_1 \vdash e_1\rho : t_2$ by induction on the premise $\Gamma, x : \mathbf{u}_1 \vdash e_1 : t_2$ (with $t_2 \simeq t$), and conclude by using [$\to$I].
[$\times$I] We have $e' \equiv (e_1\rho, e_2\rho)$. We can conclude by induction on the premises similarly to the previous case.
[$\to$E] We have $e \equiv e_1 e_2$. If $e_\circ$ is a subexpression of e_1 and/or e_2, we conclude trivially by induction (as in the previous cases).
Otherwise, $e_\circ \equiv e_1 e_2$ and thus the reduction $e_\circ \rightsquigarrow_\top e'_\circ$ uses the rule 40. Consequently, we know that $e_\circ \equiv e \equiv (\lambda x.\ e_\lambda)v$ (for some expression e_λ and value v) and $e'_\circ \equiv e' \equiv e_\lambda\{v/x\}$.
We have the following premises:
(1) $\Gamma \vdash \lambda x.\ e_\lambda : t_1 \to t_2$ (with $t_2 \simeq t$)
(2) $\Gamma \vdash v : t_1$
As the derivation is normalized, we know that the premise (1) ends with a [Inst$\wedge\leq$] pattern preceded by a [$\to$I$\wedge$] pattern (in particular, there cannot be [$\vee$] rules in between). Thus, we can extract from it a collection of derivations of the judgements $\Gamma, x : \mathbf{u}_i \vdash e_\lambda : s_i$ for $i \in I$, such that $\exists\Sigma.\ \bigwedge_{\sigma\in\Sigma}(\bigwedge_{i\in I} \mathbf{u}_i \to s_i)\sigma \leq t_1 \to t_2$ (with $\forall\sigma \in \Sigma.\ \mathrm{dom}(\sigma) \subseteq \mathcal{V}_P$). This is equivalent to $\bigwedge_{\sigma\in\Sigma}\bigwedge_{i\in I} \mathbf{u}_i \to s_i\sigma \leq t_1 \to t_2$. We define $\{(\mathbf{u}'_j, s'_j)\}_{j\in J} = \{(\mathbf{u}_i, s_i\sigma) \mid i \in I,\ \sigma \in \Sigma\}$. We thus have $\bigwedge_{j\in J} \mathbf{u}'_j \to s'_j \leq t_1 \to t_2$ (in particular, it implies $t_1 \leq \bigvee_{j\in J} \mathbf{u}'_j$).
For every $j \in J$, we know by definition that there exists σ and $i \in I$ such that $\mathbf{u}'_j = \mathbf{u}_i$ and $s'_j = s_i\sigma$. Thus, from $\Gamma, x : \mathbf{u}_i \vdash e_\lambda : s_i$, we can derive using a [Inst] rule $\Gamma, x : \mathbf{u}'_j \vdash e_\lambda : s'_j$.
Now, let's consider a partition $\{\mathbf{v}_k\}_{k\in K}$ of $\bigvee_{j\in J} \mathbf{u}'_j$ of minimal cardinality that satisfies the following property: $\forall k \in K.\ \forall j \in J.\ \mathbf{v}_k \leq \mathbf{u}'_j$ or $\mathbf{v}_k \wedge \mathbf{u}'_j \simeq \mathbb{0}$.
We can suppose that K is not empty: the case $t_1 \leq \bigvee_{j\in J} \mathbf{u}'_j \simeq \mathbb{0}$ is straightforward. For every $k \in K$, we define $J_k = \{j \in J \mid \mathbf{v}_k \wedge \mathbf{u}'_j \not\simeq \mathbb{0}\}$ (J_k cannot be empty as the partition has minimal cardinality). Note that for all $k \in K$ and $j \in J_k$, we have $\mathbf{v}_k \leq \mathbf{u}'_j$.
According to the monotonicity lemma (I.4), for every $k \in K$ and $j \in J_k$ we can derive the judgement $\Gamma, x : \mathbf{v}_k \vdash e_\lambda : s'_j$. Thus, using a [$\wedge$] rule, for every $k \in K$ we can derive $\Gamma, x : \mathbf{v}_k \vdash e_\lambda : \bigwedge_{j\in J_k} s'_j$. Moreover, as $\bigwedge_{j\in J} \mathbf{u}'_j \to s'_j \leq t_1 \to t_2$, we have for every $k \in K$: $\bigwedge_{j\in J_k} s'_j \leq t_2$. Consequently, for every $k \in K$, we can derive the judgement $\Gamma, x : \mathbf{v}_k \vdash e_\lambda : t_2$ using a [$\leq$] rule.
As v is a value and as the derivation is normalized, we can apply I.14 on the premise (2) and deduce that t_1 cannot be decomposed into a non-trivial union. As $\{\mathbf{v}_k\}_{k\in K}$ is a partition covering t_1, we can therefore find $k \in K$ such that $t_1 \simeq \mathbf{v}_k \wedge t_1$. Thus, from $\Gamma, x : \mathbf{v}_k \vdash e_\lambda : t_2$, we can easily derive $\Gamma \vdash e_\lambda\{v/x\} : t_2$ by replacing [Ax$_\lambda$] rules by the derivation (1) and using monotonicity (I.4).
[$\times$E$_1$] We have $e \equiv \pi_1 e_1$. If $e_\circ$ is a subexpression of e_1, we conclude trivially by induction.

Otherwise, the reduction $e_\circ \rightsquigarrow_\top e'_\circ$ uses the rule 41 and thus we know that $e_\circ \equiv e \equiv \pi_1(v_1, v_2)$ and $e'_\circ \equiv e' \equiv v_1$.

Similarly to the previous case, as the derivation is normalized, we can extract from the derivation of the premise $\Gamma \vdash (v_1, v_2) : t_1 \times t_2$ a collection of derivations of the judgements $\Gamma \vdash v_1 : s_i$ and $\Gamma \vdash v_2 : s'_i$ for $i \in I$, such that $\exists \Sigma.\ \bigwedge_{\sigma\in\Sigma} \bigwedge_{i\in I} (s_i \times s'_i)\sigma \leq t_1 \times t_2$. In particular, this last property implies $\bigwedge_{\sigma\in\Sigma} \bigwedge_{i\in I} s_i\sigma \leq t_1$.

Therefore, we can conclude this case by using a [Inst$\wedge\leq$] pattern with the premises $\{\Gamma \vdash v_1 : s_i\}_{i\in I}$ in order to derive $\Gamma \vdash v_1 : t_1$.

[$\times$**E**$_2$] Similar to the previous case.

[$\vee$] We have $e \equiv e_1\{e_2/\mathsf{x}\}$ (conclusion of the [$\vee$] rule), and thus $e' \equiv (e_1\{e_2/\mathsf{x}\})\{e'_\circ/e_\circ\}$.

If e_2 is a value, then we can apply I.14 on the first premise $\Gamma \vdash e_2 : s$ (as the derivation is normalised), which gives that there exists $i \in I$ such that $s \wedge \mathbf{u}_i \simeq s$. The corresponding premise, $\Gamma, \mathsf{x} : s \vdash e_1 : t$, can be transformed into a derivation $\Gamma \vdash e_1\{e_2/\mathsf{x}\} : t$ by replacing any occurrence of x by e_2 and by replacing occurrences of an [Ax$_\vee$] rule on x by the derivation of the first premise, $\Gamma \vdash e_2 : s$ (where Γ can be extended as needed to match the current environment). Then, we conclude by applying the induction hypothesis on this new derivation.

We can now assume that e_2 is not a value. We know that $e_\circ$ does not contain x as a free variable (otherwise there would be no occurrence of $e_\circ$ in $e_1\{e_2/\mathsf{x}\}$). Moreover, $e'_\circ$ does not contain x neither, because a reduction step cannot introduce a new free variable.

There are several cases:

- $e_\circ$ does not contain e_2 and e_2 does not contain $e_\circ$. In this case, we have: $e' \equiv (e_1\{e_2/\mathsf{x}\})\{e'_\circ/e_\circ\} \equiv (e_1\{e'_\circ/e_\circ\})\{e_2/\mathsf{x}\}$. Thus, we can easily conclude by keeping the first premise of the [$\vee$] rule and applying the induction hypothesis on the others.
- e_2 contains $e_\circ$. In this case, we pose $e'_2 = e_2\{e'_\circ/e_\circ\}$. We have $e' \equiv (e_1\{e_2/\mathsf{x}\})\{e'_\circ/e_\circ\} \equiv (e_1\{e'_\circ/e_\circ\})\{e'_2/\mathsf{x}\}$. We can easily derive $\Gamma \vdash e'_2 : s$ by induction on the first premise, and $\Gamma, \mathsf{x} : s \wedge \mathbf{u}_i \vdash e_1\{e'_\circ/e_\circ\} : t$ for all $i \in I$ by induction on the others. Thus, we can derive $\Gamma \vdash (e_1\{e'_\circ/e_\circ\})\{e'_2/\mathsf{x}\} : t$ using a [$\vee$] rule.
- $e_\circ$ contains e_2 as a strict subexpression. In this case, we pose $e_\bullet = e_\circ\{\mathsf{x}/e_2\}$ and $e'_\bullet = e'_\circ\{\mathsf{x}/e_2\}$. We know that e_1 does not contain any occurrence of e_2 (because the derivation is normalised), and thus no occurrence of $e_\circ$ neither. Consequently, we have $e' \equiv (e_1\{e_2/\mathsf{x}\})\{e'_\circ/e_\circ\} \equiv (e_1\{e'_\bullet/e_\bullet\})\{e_2/\mathsf{x}\}$.
  As e_2 is not a value, it can only appear in $e_\circ$ inside a lambda-abstraction, and/or inside a branch of a typecase: otherwise, e_2 would necessarily be a value for $e_\circ$ to be reducible. Thus, we can deduce that $e_\bullet = e_\circ\{\mathsf{x}/e_2\} \rightsquigarrow_\top e'_\circ\{\mathsf{x}/e_2\} = e'_\bullet$. Consequently, we can easily conclude by keeping the first premise of the [$\vee$] rule and applying the induction hypothesis on the others.

[$\mathbb{0}$] We have $e \equiv (e_1{\in}\tau)\ ?\ e_2 : e_3$. As values cannot have the type $\mathbb{0}$, we know that e_1 is not a value. Thus, $e' \equiv (e_1\rho{\in}\tau)\ ?\ e_2\rho : e_3\rho$. We can derive $\Gamma \vdash e_1\rho : \mathbb{0}$ by induction on the premise, and then we can derive $\Gamma \vdash e' : \mathbb{0}$ by using [$\mathbb{0}$].

[$\in_1$] We have $e \equiv (e_1{\in}\tau)\ ?\ e_2 : e_3$. There are three cases:

$e' \equiv (e_1\rho{\in}\tau)\ ?\ e_2\rho : e_3\rho$ We can easily conclude by induction on the premises.

$e' \equiv e_2$ We can conclude with the second premise.

$e' \equiv e_3$ This case is impossible. Indeed, it implies that e_1 is a value, and as $\Gamma \vdash e_1 : \tau$ (first premise), we can deduce using the property I.13 that $e_1 \in \tau$, which contradicts $e \rightsquigarrow_\top e_3$.

[$\in_2$] Similar to the previous case.

□

Corollary I.16 (Subject reduction). *If* $\Gamma \vdash e : t$ *and* $e \rightsquigarrow_{\mathcal{P}} e'$, *then* $\Gamma \vdash e' : t$.

Proof. The root of the derivation of $e \rightsquigarrow_{\mathcal{P}} e'$ is a $[\kappa]$ rule, with its premise being of the form $e_\circ \rightsquigarrow_{\top} e'_\circ$. Additionally, we have $e' \equiv e\{e'_\circ/e_\circ\}$. Thus, by using I.15, we obtain $\Gamma \vdash e' : t$ □

I.1.5 Progress.

Theorem I.17 (Progress). *If* $\Gamma \vdash e : t$ *and if there is no evaluation context* E *and variable* x *(resp.* $\mathtt{x}$*) such that* $e \equiv E[x]$ *(resp.* $e \equiv E[\mathtt{x}]$*), then either* e *is a value or* $\exists e'. \; e \rightsquigarrow_{\mathcal{P}} e'$.

Proof. For convenience, quantifications on a variable x (like in $\forall E, x. \; e \not\equiv E[x]$) will denote both lambda variables and binding variables.

We apply the normalisation lemma (I.11) to the derivation of the judgement $\Gamma \vdash e : t$, and we proceed by induction on $(n_{\vee_\top}, n)$ (using the lexicographic order), with $n_{\vee_\top}$ the number of top-level $[\vee]$ rules in the derivation (i.e. that are not in the subderivation of a $[\rightarrow$I$]$ rule), and n the total number of rules in the derivation.

Note that, even if the derivation is initially normalized, some binding variables will be substituted by values in some inductive calls, thus the properties of I.7 about structural rules will not hold. The other properties of normalisation, however, will be preserved.

We consider the root of the derivation:

[Const] Trivial (e is a value).
[Ax$_\lambda$] Impossible case (e cannot be a variable).
[Ax$_\vee$] Impossible case (e cannot be a variable).
[$\leq$] Trivial (by induction on the premise).
[Inst] Trivial (by induction on the premise).
[$\wedge$] Trivial (by induction on one of the premises).
[$\rightarrow$I] Trivial (e is a value).
[$\times$I] We have $e \equiv (e_1, e_2)$.

- If e_1 is not a value, and as we have $\forall E, x. \; e_1 \not\equiv E[x]$, we know by applying the induction hypothesis that e_1 can be reduced. Thus, e can also be reduced with the context $([\,], e_2)$.
- If e_1 is a value, then we can apply the induction hypothesis on the second premise (as e_1 is a value, we know that $\forall E, x. \; e_2 \not\equiv E[x]$). It gives that either e_2 is a value or it can be reduced. We can easily conclude in both cases (if e_2 is a value, then e is also a value, otherwise, e can be reduced with the context $(e_1, [\,])$).

[$\rightarrow$E] We have $e \equiv e_1 e_2$, with $\Gamma \vdash e_1 : s \rightarrow t$ and $\Gamma \vdash e_2 : s$.

- If e_1 is not a value, and as we have $\forall E, x. \; e_1 \not\equiv E[x]$, we know by applying the induction hypothesis that e_1 can be reduced. Thus, e can also be reduced with the context $[\,]e_2$.
- If e_1 is a value, we can apply the property I.13 on it. As $\Gamma \vdash e_1 : \mathbb{0} \rightarrow \mathbb{1}$, it gives that $e_1 \in \mathbb{0} \rightarrow \mathbb{1}$ and thus $e_1 \equiv \lambda x. \; e_\circ$. Moreover, we can apply the induction hypothesis on the second premise (as e_1 is a value, we know that $\forall E, x. \; e_2 \not\equiv E[x]$). It gives that either e_2 is a value or it can be reduced. We can easily conclude in both cases (if e_2 is a value, then e can be reduced using the rule 40, otherwise, e can be reduced with the context $e_1[\,]$).

[$\times$E$_1$] We have $e \equiv \pi_1 e_\circ$, with $\Gamma \vdash e_\circ : t \times s$. By induction on the premise, we know that $e_\circ$ is either a value or it can be reduced. If $e_\circ$ can be reduced, then e can also be reduced with the context $\pi_1[\,]$. Otherwise, as $\Gamma \vdash e_\circ : \mathbb{1} \times \mathbb{1}$, we know by using the property I.13 that $e_\circ \in \mathbb{1} \times \mathbb{1}$. Thus, $e_\circ \equiv (v_1, v_2)$ (with v_1 and v_2 two values) and consequently e can be reduced using the rule 41.
[$\times$E$_2$] Similar to the previous case.
[$\vee$] We have $e \equiv e_1\{e_2/\mathtt{x}\}$, with $\Gamma \vdash e_2 : s$ and $\forall i \in I. \; \Gamma, \mathtt{x} : s \wedge \mathbf{u}_i \vdash e_1 : t$. There are two cases:

- There exists a reduction context E such that $e_1 \equiv E[\mathrm{x}]$. In this case, we know that $\forall E', y.\ e_2 \not\equiv E'[y]$, otherwise we would have $e \equiv E[E'[y]]$. Thus, we can apply the induction hypothesis on the first premise. It gives that either e_2 is a value or it can be reduced.
  If e_2 can be reduced, then e can also be reduced with the context E.
  Otherwise, e_2 is a value, thus we can apply I.14 on the first premise $\Gamma \vdash e_2 : s$ (as the derivation is normalised), which gives that there exists $i \in I$ such that $s \wedge \mathbf{u}_i \simeq s$. The corresponding premise, $\Gamma, \mathrm{x} : s \wedge \mathbf{u}_i \vdash e_1 : t$, can be transformed into a derivation $\Gamma \vdash e_1\{e_2/\mathrm{x}\} : t$ by replacing any occurrence of x by e_2 and by replacing occurrences of an $[\mathrm{Ax}_\vee]$ rule on x by the derivation of the first premise, $\Gamma \vdash e_2 : s$ (where Γ can be extended as needed to match the current environment). Then, we conclude by applying the induction hypothesis on this new derivation.
- There is no reduction context E such that $e_1 \equiv E[\mathrm{x}]$. Thus, we can apply the induction hypothesis on the n-th premises ($n \geq 2$). It gives that either e_1 is a value or it can be reduced. We can easily conclude in both cases (if e_1 is a value, then e is also a value, and if e_1 can be reduced, then e can also be reduced).

$[\mathbb{0}]$ We have $e \equiv (e_\circ{\in}\tau)\,?\,e_1 : e_2$, with $\Gamma \vdash e_\circ : \mathbb{0}$. As values cannot have the type $\mathbb{0}$, we know that $e_\circ$ is not a value. Thus, by induction on the premise, we know that $e_\circ$ can be reduced. Consequently, e can be reduced with the context $([\,]{\in}\tau)\,?\,e_1 : e_2$.

$[\in_1]$ We have $e \equiv (e_\circ{\in}\tau)\,?\,e_1 : e_2$, with $\Gamma \vdash e_\circ : \tau$. By induction on the first premise, we know that $e_\circ$ is either a value or it can be reduced. If $e_\circ$ is a value, then e can be reduced using the rule 43. Otherwise, $e_\circ$ can be reduced and thus e can also be reduced with the context $([\,]{\in}\tau)\,?\,e_1 : e_2$.

$[\in_2]$ Similar to the previous case.

□

Corollary I.18 (Progress). *If $\varnothing \vdash e : t$, then either e is a value or $\exists e'.\ e \rightsquigarrow_{\mathcal{P}} e'$.*

Proof. We can deduce from $\varnothing \vdash e : t$ that there is no evaluation context E and variable x (resp. x) such that $e \equiv E[x]$ (resp. $e \equiv E[\mathrm{x}]$). Thus, we can conclude with I.17. □

Property I.19. *For any e and v, if $e \rightsquigarrow_{\mathcal{P}}^* v$ or $e \rightsquigarrow_{\mathcal{P}}^*$ (e diverges with $\rightsquigarrow_{\mathcal{P}}$), then either there exists v' such that $e \rightsquigarrow^* v'$ or $e \rightsquigarrow^*$ (e diverges with $\rightsquigarrow$). (see Appendix B for the definition of the semantics $\rightsquigarrow$)*

Proof. We define the following syntax for expressing a path in an expression:

$$\textbf{Path} \quad \phi ::= [\,] \mid \phi\_ \mid \_\phi \mid (\phi, \_) \mid (\_, \phi) \mid \lambda\phi \mid (\phi{\in}\_)\,?\,\_ : \_ \mid (\_{\in}\_)\,?\,\phi : \_ \mid (\_{\in}\_)\,?\,\_ : \phi$$

We first introduce a new semantics $\rightsquigarrow_C$ similar to $\rightsquigarrow_\top$ but where the reductions can happen under any context (not just an evaluation context).

We trivially have that following proprerty: for any e and v, if $e \rightsquigarrow_{\mathcal{P}}^* v$ then $e \rightsquigarrow_C^* v$. Thus, we only need to prove the following property: for any e, if $e \rightsquigarrow_C^* v$ (for any v) or $e \rightsquigarrow_C^*$, then $e \rightsquigarrow^*$ or there exists v' such that $e \rightsquigarrow^* v'$.

We will assume that $e \rightsquigarrow_C^* v$ (for some v) or $e \rightsquigarrow_C^*$, and show that either e is a value or $e \rightsquigarrow e'$ for some e' such that $e' \rightsquigarrow_C^* v'$ (for some v') or $e' \rightsquigarrow_C^*$. The result above can then easily be deduced by iterating.

Let's represent each reduction happening in $e \rightsquigarrow_C^* v$ or $e \rightsquigarrow_C^*$ by the path under which it is happening (the associated top-level reduction can easily be retrieved as at most one top-level reduction can apply on a given expression). It gives us a (potentially infinite) list of paths. Now, let's consider the first path in this list (if any) that corresponds to a valid reduction context in e (i.e. such that the _ in the path can be completed to give a reduction context matching e):

- If there is no such path, and as the final expression is a value, we know that all these reductions happened inside a lambda. Thus, e is also a value.
- If such a path ϕ exists, then we know that all the others reductions in the list before it are happening on a path that is not a prefix of ϕ (because evaluation contexts are closed by inclusion). Thus, the path ϕ must exist in e. Let's call e_ϕ the subexpression at the path ϕ in e, and e'_ϕ the actual subexpression that was reduced in the initial reduction sequence $e \rightsquigarrow_C^* v$ or $e \rightsquigarrow_C^*$.
  We know that e'_ϕ is reducible, and we also know that e'_ϕ can be obtained from e_ϕ only by performing reductions not in an evaluation context (as the path we choosed is the first to be a reduction context). We can deduce that e_ϕ is also reducible.
  Thus, in our list of paths representing the reduction sequence $e \rightsquigarrow_C^* v$ or $e \rightsquigarrow_C^*$, we can move the first occurrence of ϕ in first position and update the paths that were before it (and that were suffixes of ϕ) in order to obtain the same expression as before (some reductions might need to be removed or duplicated). We obtain a sequence of reductions $e \rightsquigarrow_C e' \rightsquigarrow_C^* v$ or $e \rightsquigarrow_C e' \rightsquigarrow_C^*$ such that $e \rightsquigarrow e'$, which concludes the proof.

□

Theorem I.20 (Type safety). *For any expression e, if $\varnothing \vdash e : t$, then either $e \rightsquigarrow^* v$ with $\varnothing \vdash v : t$ or $e \rightsquigarrow^*$ (e diverges).*

Proof. Straightforward application of I.16, I.18 and I.19. □

I.2 Algorithmic type system

I.2.1 Maximal Sharing Canonical Form. As defined in I.1.1, we will consider that expressions of the source language can contain binding variables. Consequently, the unwinding operator $\lceil . \rceil$ can freely be used on atoms and on canonical forms containing free binding variables.

Property I.21. *For any expression e of the source language, $\lceil \mathsf{term}([\![e]\!]) \rceil \equiv_\alpha e$.*

Proof. Straightforward structural induction on e. □

Property I.22. *If $\kappa \equiv_\kappa \kappa'$, then $\lceil \kappa \rceil \equiv_\alpha \lceil \kappa' \rceil$.*

Proof. If a reordering as defined in Definition 3.1 applies at top-level on the expression $\mathsf{bind}\, \mathsf{x}_1 = a_1 \,\mathsf{in}\, \mathsf{bind}\, \mathsf{x}_2 = a_2 \,\mathsf{in}\, \kappa$, the unwinding remains unchanged: as $\mathsf{x}_1 \notin \mathsf{fv}(a_2)$ and $x_2 \notin \mathsf{fv}(a_1)$, we have $\kappa\{a_1/\mathsf{x}_1\}\{a_2/\mathsf{x}_2\} = \kappa\{a_2/\mathsf{x}_2\}\{a_1/\mathsf{x}_1\}$.

The general case can easily be deduced with the observation that $\forall C, \kappa_1, \kappa_2.\ \lceil \kappa_1 \rceil \equiv_\alpha \lceil \kappa_2 \rceil \Rightarrow \lceil C[\kappa_1] \rceil \equiv_\alpha \lceil C[\kappa_2] \rceil$ (with C denoting an arbitrary context). □

Property I.23 (Equivalence of MSC-forms). *If κ_1 and κ_2 are two MSC-forms and $\lceil \kappa_1 \rceil \equiv_\alpha \lceil \kappa_2 \rceil$, then $\kappa_1 \equiv_\kappa \kappa_2$.*

Proof. We will show that κ_2 can be transformed into κ_1 just with alpha-renaming and reordering of independent bindings (as specified in the definition of $\equiv_\kappa$).

We represent κ_1 as a pair (b_1, e_1) with a b_1 being a list of definitions representing its top-level bindings, and e_1 its final binding variable (noted e_1 because we allow any expression in order to be more general).

More formally, it gives a representation (b, e) using the following syntax:

$$\textbf{List of definitions} \quad b \quad ::= \quad \mathsf{x} \mapsto a; \ldots; \mathsf{x} \mapsto a \tag{45}$$

Similarly, we represent κ_2 as a pair (b_2, e_2).

We can define an unwinding operator $\lceil . \rceil$, similar to the one defined in Appendix E, that transforms this representation into an expression of the source language. As $\lceil \kappa_1 \rceil \equiv_\alpha \lceil \kappa_2 \rceil$, we have $\lceil (b_1, e_1) \rceil \equiv_\alpha \lceil (b_2, e_2) \rceil$. We alpha-rename (b_2, e_2) so that $e_2 = e_1$ and $\lceil (b_1, e) \rceil = \lceil (b_2, e) \rceil$.

Now, let's prove the following property (from which the property to prove can be deduced): let b_1 and b_2 be two lists of definitions and e be an expression such that:

- $\lceil (b_1, e) \rceil = \lceil (b_2, e) \rceil$
- The body of lambdas in b_1 and b_2 are in MSC-form (3.2)
- Both b_1 and b_2 respect the following properties (corresponding to the MSC-form properties applied to the top-level definitions), written here for a list of definitions b:
  (1) if $\mathsf{x}_1 \mapsto a_1$ and $\mathsf{x}_2 \mapsto a_2$ are distinct definitions in b, then $a_1 \not\equiv_\kappa a_2$
  (2) for any definition $\mathsf{x} \mapsto \lambda z.\kappa$ in b, any binding $\mathtt{bind}\, \mathsf{y} = a \,\mathtt{in}\, \kappa'$ in κ is such that $\mathrm{fv}(a) \not\subseteq \mathrm{fv}(\lambda z.\kappa)$
  (3) if b contains a definition $\mathsf{x} \mapsto a$, then x is a free variable of one of the next definitions or of e

Then, we can transform b_2 into b_1 just with alpha-renaming, reordering of independent definitions, and replacement of an atom by a $\equiv_\kappa$-equivalent one.

We prove this property by induction on the number of definitions in b_1 + the total number of bindings in the atoms.

The base case ($b_1 = \varnothing$) is trivial: as $\lceil (b_2, e) \rceil = \lceil (b_1, e) \rceil = \lceil (\varnothing, e) \rceil = e$, we deduce with property 3 that $b_2 = \varnothing$.

For the inductive case, let's say $b_1 = (b_1'; \mathsf{x} \mapsto a_1)$.

With property 3, we know that x appears in e. As $\lceil (b_1, e) \rceil = \lceil (b_2, e) \rceil$, the binding variable x that appears in e must be unwinded to the same subexpression using the definitions in b_1 than using the definitions in b_2. Thus, b_2 must also feature a definition for x, let's call a_2 the associated atom. We move in b_2 the definition $\mathsf{x} \mapsto a_2$ at the end (if not already), it gives $b_2 = (b_2'; \mathsf{x} \mapsto a_2)$. We then have $\lceil (b_1', a_1) \rceil = \lceil (b_2', a_2) \rceil$. As every kind of atom introduces a different syntactic construction, we can deduce that a_1 and a_2 are atoms of the same kind.

- If a_1 and a_2 are atoms that are not lambdas and that do not contain any binding variable (constants, lambda variables), we directly have $a_1 = a_2$.
- If a_1 and a_2 are atoms that are not lambdas and that contain only one binding variable (projections), we can alpha-rename binding variables in b_2 so that $a_1 = a_2$.
- If a_1 and a_2 are atoms that are not lambdas and that contain two binding variables (applications, pairs), we consider two cases:
  - If, in $\lceil (b_1', a_1) \rceil (= \lceil (b_2', a_2) \rceil)$, these two binding variables are unwinded to the same expressions (modulo alpha renaming), we do the following. Let's call x and y the two binding variables in a_1, and let's show that we necessarily have $\mathsf{x} = \mathsf{y}$. We consider the list of definitions b_x, which is a cleaned version of b_1' where all the definitions that are not related (directly or indirectly) to x_1 have been removed. Similarly, we consider the list of definitions b_y where the definitions unrelated to y have been removed. Then, we apply the induction hypothesis to the lists of definition b_x, $b_\mathsf{y}\{\mathsf{x}/\mathsf{y}\}$ and the expression x. This tells us that b_x and b_y are equivalent modulo reordering of the definitions, alpha-renaming and $\equiv_\kappa$-transformation of atoms. Thus, according to the property 1, x and y cannot have two distinct definitions in b_1', and thus $\mathsf{x} = \mathsf{y}$. The same reasoning

can be done for a_2, and thus we deduce that both a_1 and a_2 contain the same binding variable twice. Thus, we can alpha-rename binding variables in b_2 so that $a_1 = a_2$.
  - Otherwise, the two binding variables in a_1 must be different, and the same applies to a_2. Thus, we can alpha-rename binding variables in b_2 so that $a_1 = a_2$.
- If a_1 and a_2 are typecases (containing 3 binding variables), we proceed similarly to the previous case to obtain $a_1 = a_2$.
- In the case where a_1 and a_2 are lambdas, let's say $\lambda x.\ \kappa_1$ and $\lambda x.\ \kappa_2$ respectively, we note $(b_{\mathsf{x}_1}, \mathsf{x}_1)$ and $(b_{\mathsf{x}_2}, \mathsf{x}_2)$ the representations of κ_1 and κ_2 respectively. We now consider the representation $(b'_1; b_{\mathsf{x}_1}, \lambda x.\ \mathsf{x}_1)$ and remove from it all the unused definitions (i.e. not related to x_1), it gives us a new representation $(b''_1, \lambda x.\ \mathsf{x}_1)$. We do the same for $(b'_2; b_{\mathsf{x}_2}, \lambda x.\ \mathsf{x}_2)$, it gives a new representation $(b''_2, \lambda x.\ \mathsf{x}_2)$. By applying the alpha-renaming $\{\mathsf{x}_1/\mathsf{x}_2\}$ to b_2, we can manage to get $\lambda x.\ \mathsf{x}_2 = \lambda x.\ \mathsf{x}_1 = e''$, and we call the induction hypothesis on b''_1, b''_2 and e''. It gives us that b''_1 and b''_2 are equivalent modulo reordering of the definitions, alpha-renaming and $\equiv_\kappa$-transformation of the atoms. By using the property 2, we can deduce that the same applies to b_{x_1} and b_{x_2}. Thus, we can alpha-rename b_2 and $\equiv_\kappa$-transform some of its atoms so that $b_{\mathsf{x}_1} = b_{\mathsf{x}_2}$, and thus $a_1 = a_2$.

In any case, we get $a_1 = a_2 = a$, thus the last definition of b_1 is the same as the last definiton of b_2. The same can be proven for the previous definitions by using the induction hypothesis on b'_1, b'_2 and $e\{a/\mathsf{x}\}$. □

Property I.24. *If $\kappa \dashrightarrow \kappa'$, then $\lceil \kappa \rceil \equiv_\alpha \lceil \kappa' \rceil$.*

Proof. Straightforward: this proof is similar to (and uses) the proof of I.22. □

Property I.25 (Normalisation). *There is no infinite chain $\kappa_1 \dashrightarrow \kappa_2 \dashrightarrow \cdots$*

Proof. Let n be the maximal number of nested lambdas in κ_1. We call depth of a binding the number of nested lambdas it is into (the depth of a binding of κ_1 is at most n).

Let $N_\kappa(i)$ be the number of bindings of depth i in an expression κ. Let $S(\kappa)$ be the following n-uplet: $(N_\kappa(n), N_\kappa(n-1), \ldots, N_\kappa(0))$.

The chain $S(\kappa_1), S(\kappa_2), \ldots$ is strictly decreasing with respects to the lexical order, thus it cannot be infinite. □

Property I.26. *If $\kappa \not\dashrightarrow$, then κ is an MSC-form.*

Proof. Let's assume we have $\kappa \not\dashrightarrow$ and show that all 3 MSC properties are satisfied.

The property 3 (removing of unused bindings) is trivial: any binding that does not satisfy this property can directly be eliminated with the rule 36. As the rule 36 does not apply, this property must be satisfied.

Now, let's focus on the property 2 (extrusion of bindings). We assume that there exists a subexpression $\lambda x.\ \kappa_1$ of κ and a subexpression $\mathtt{bind}\, \mathsf{y} = a\, \mathtt{in}\, \kappa_2$ of κ_1 such that $\mathsf{fv}(a) \subseteq \mathsf{fv}(\lambda x.\ \kappa_1)$. We know that the definition a cannot depend on any variable defined inside $\lambda x.\ \kappa_1$ (including x), otherwise this variable would be in $\mathsf{fv}(a)$ and not in $\mathsf{fv}(\lambda x.\ \kappa_1)$. Thus, we can reorder the binding y (38) in the first position of its inner-most containing lambda-abstraction, and then apply the rule 37 on it, which contradicts $\kappa \not\dashrightarrow$. Thus, the property 2 is satisfied.

Finally, let's show that the property 1 (sharing of equivalent definitions) is satisfied too. We assume that there are two distinct bindings $\mathtt{bind}\, \mathsf{x}_1 = a_1\, \mathtt{in}\, \ldots$ and $\mathtt{bind}\, \mathsf{x}_2 = a_2\, \mathtt{in}\, \ldots$ such that $a_1 \equiv_\kappa a_2$. As the property 2 is satisfied, and as $\mathsf{fv}(a_1) = \mathsf{fv}(a_2)$, we know that these two bindings are on the same level (i.e. their contexts cross exactly the same lambda-abstractions). Thus, we can reorder them (38) to be the one next to the other so that the rule 35 is applicable, which contradicts $\kappa \not\dashrightarrow$. Thus, the property 1 is satisfied. □

Property I.27 (Confluence). *Let κ_1, κ_2 and κ_2' such that $\kappa_1 \dashrightarrow^* \kappa_2$ and $\kappa_1 \dashrightarrow^* \kappa_2'$. Then, there exists κ_3 and κ_3' such that $\kappa_2 \dashrightarrow^* \kappa_3$, $\kappa_2' \dashrightarrow^* \kappa_3'$ and $\kappa_3 \equiv_\kappa \kappa_3'$.*

Proof. Immediate corollary of I.25 (normalisation), I.24 (preservation of $\lceil . \rceil$), I.26 ($\not\dashrightarrow$ implies MSC-form) and I.23 (equivalence of MSC-forms). □

I.2.2 Soundness. See Appendix G for the full algorithmic system, without the rules for extensions.

Lemma I.28. *If κ is a canonical form where all the binding variables introduced are distinct, and if $\Gamma \vdash_{\mathcal{A}} [\kappa \mid \Bbbk] : t$, then the associated derivation does not need to perform any implicit alpha-renaming on binding variables.*

Proof. Straightforward induction (note that it relies on the guardian $\mathrm{x} \notin \mathrm{dom}(\Gamma)$ of the [Bind$_1$-Alg] rule). □

Thus, in the following, we assume that all the binding variables introduced by a canonical form are distinct, and that derivations of the algorithmic type system never perform implicit alpha-renaming on them.

Theorem I.29 (Soundness).
If $\Gamma \vdash_{\mathcal{A}} [\kappa \mid \Bbbk] : t$, then $\Gamma \vdash \lceil \kappa \rceil : t$. If $\Gamma \vdash_{\mathcal{A}} [a \mid \mathbb{a}] : t$, then $\Gamma \vdash \lceil a \rceil : t$.

Proof. We proceed by structural induction on the typing derivation.
We consider the root of the derivation:

[Const-Alg] Trivial ([Const] rule).

[Ax-Alg] Trivial ([Ax$_\lambda$] rule).

[→I-Alg] We have $a = \lambda x.\ \kappa$, and thus $\lceil a \rceil = \lambda x.\ \lceil \kappa \rceil$.
By induction on the premise, we get $\Gamma, x : \mathbf{u} \vdash \lceil \kappa \rceil : s$. By applying the rule [→I], we get $\Gamma \vdash \lceil a \rceil : \mathbf{u} \to s$.

[→E-Alg] We have $a = \mathrm{x}_1\mathrm{x}_2$. We pose $t_1 = \Gamma(\mathrm{x}_1)\Sigma_1$ and $t_2 = \Gamma(\mathrm{x}_2)\Sigma_2$.
With an [Ax$_\vee$] rule, we can derive $\Gamma \vdash \mathrm{x}_1 : \Gamma(\mathrm{x}_1)$ and $\Gamma \vdash \mathrm{x}_2 : \Gamma(\mathrm{x}_2)$. By using a [Inst∧≤] pattern, we can derive from that $\Gamma \vdash \mathrm{x}_1 : t_1$ and $\Gamma \vdash \mathrm{x}_2 : t_2$. We have $t \simeq t_1 \circ t_2$. Thus, according to the definition of $\circ$, we know that $t_1 \leq t_2 \to t$. Thus, with an application of [≤] on $\Gamma \vdash \mathrm{x}_1 : t_1$, we can derive $\Gamma \vdash \mathrm{x}_1 : t_2 \to t$. We can then conclude with an application of the rule [→E].

[×I-Alg] We have $a = (\mathrm{x}_1, \mathrm{x}_2)$.
With an [Ax$_\vee$] rule, we can derive $\Gamma \vdash \mathrm{x}_1 : \Gamma(\mathrm{x}_1)\rho_1$ and $\Gamma \vdash \mathrm{x}_2 : \Gamma(\mathrm{x}_2)\rho_2$ (with ρ_1 and ρ_2 as in the [×I-Alg] rule). We can then conclude with an application of the rule [×I].

[×E$_1$-Alg] We have $a = \pi_1\mathrm{x}$. We pose $t_\circ = \Gamma(\mathrm{x})\Sigma$.
With an [Ax$_\vee$] rule, we can derive $\Gamma \vdash \mathrm{x} : \Gamma(\mathrm{x})$. By using a [Inst∧≤] pattern, we can derive from that $\Gamma \vdash \mathrm{x} : t_\circ$. We have $t \simeq \boldsymbol{\pi}_1 t_\circ$. Thus, according to the definition of $\boldsymbol{\pi}_1$, we can deduce that $t_\circ \leq t \times \mathbb{1}$. Thus, with an application of [≤], we can derive $\Gamma \vdash \mathrm{x} : t \times \mathbb{1}$. We can then conclude with an application of the rule [×E$_1$].

[×E$_2$-Alg] Similar to the previous case.

[$\mathbb{0}$-Alg] Similar to the previous case.

[$\in_1$-Alg] Similar to the previous case.

[$\in_2$-Alg] Similar to the previous case.

[Var-Alg] Trivial ([Ax$_\vee$] rule).

[Bind$_1$-Alg] We have $\kappa = \mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\,\kappa_\circ$ and thus $\lceil \kappa \rceil = \lceil \kappa_\circ \rceil\{\lceil a \rceil/\mathrm{x}\}$.
By induction on the premise, we get $\Gamma \vdash \lceil \kappa_\circ \rceil : t$. As $\mathrm{x} \notin \mathrm{dom}(\Gamma)$, we know that this derivation does not contain any [Ax$_\vee$] rule applied on x. We can thus easily transform it into a derivation $\Gamma \vdash \lceil \kappa_\circ \rceil\{\lceil a \rceil/\mathrm{x}\} : t$ just be replacing every occurrence of x by $\lceil a \rceil$.

[Bind$_2$-Alg] We have $\kappa = \mathtt{bind}\, \mathtt{x} = a\, \mathtt{in}\, \kappa_\circ$ and thus $\lceil\kappa\rceil = \lceil\kappa_\circ\rceil\{\lceil a\rceil/\mathtt{x}\}$.
By induction on the first premise, we get $\Gamma \vdash \lceil a\rceil : s$. For any $i \in I$ ($I \neq \varnothing$), we apply the induction hypothesis on the corresponding premise. It gives $\Gamma, \mathtt{x} : s \wedge \mathbf{u}_i \vdash \lceil\kappa_\circ\rceil : t_i$. With a $[\leq]$ rule, we can obtain $\Gamma, \mathtt{x} : s \wedge \mathbf{u}_i \vdash \lceil\kappa_\circ\rceil : t$ (with $t \simeq \bigvee_{i\in I} t_i$). Thus, we can conclude using a $[\vee]$ rule.

[$\wedge$-Alg] Trivial induction on the premises.

□

I.2.3 Completeness. In the following, we fix a total order $\leq$ over the expressions, compatible with the subexpression order (as in I.1.2). For any expression e, it determines a unique MSC-form $\mathsf{MSC}(e)$: independent consecutive bindings can be ordered depending on the order $\leq$ of their unwinding. Note that all MSC-forms of a given expression are equivalent modulo $\equiv_\kappa$ (I.23), so the order $\leq$ is only a way to fix the order of independent bindings for convenience. As the order taken is arbitrary, the proofs below will work for any MSC-form.

Lemma I.30 (Monotonicity).
If $\Gamma \vdash_{\mathcal{A}} [\kappa \mid \mathbb{k}] : t$ and $\Gamma' \leq_P \Gamma$, then $\exists \mathbb{k}', t'.\ \Gamma' \vdash_{\mathcal{A}} [\kappa \mid \mathbb{k}'] : t'$ with $t' \leq_P t$.
If $\Gamma \vdash_{\mathcal{A}} [a \mid \mathbb{a}] : t$ and $\Gamma' \leq_P \Gamma$, then $\exists \mathbb{a}', t'.\ \Gamma' \vdash_{\mathcal{A}} [a \mid \mathbb{a}'] : t'$ with $t' \leq_P t$.

Proof. Straightforward induction on the derivation. □

Definition I.31 (Form derivation). *A form derivation is a derivation of the declarative system such that:*

- *It satisfies the properties of the normalisation lemma I.11,*
- *Either it does not contain any structural rule, or its root is a $[\vee]$ rule (or a [Inst$\wedge\leq$] pattern with a $[\vee]$ rule as premise)*

Definition I.32 (Atom derivation). *An atom derivation is a derivation of the declarative system such that:*

- *It satisfies the properties of the normalisation lemma I.11,*
- *It contains at least one structural rule,*
- *It has no occurrence of a $[\vee]$ rule except in the subderivation of a $[\rightarrow I]$ rule,*
- *Its root is not a [Inst] nor a $[\leq]$*

Definition I.33 (Atomic source expression). *We say that an expression of the source language is an atomic source expression if it has the following shape:*

$$\textbf{\textit{Atomic source expressions}} \quad \bar{e} \quad ::= \quad c \mid x \mid \lambda x.e \mid (\mathtt{x},\mathtt{x}) \mid \mathtt{x}\mathtt{x} \mid \pi_i\mathtt{x} \mid (\mathtt{x}{\in}\tau)\,?\,\mathtt{x}:\mathtt{x} \qquad (46)$$

and such that, for the case $\lambda x.e$, all subexpressions of e are either a binding variable or they contain a lambda variable that is not in $\mathsf{fv}(\lambda x.e)$.
The variable $\bar{e}$ is used to range over atomic source expressions.

Definition I.34. *For any atomic source expression $\bar{e}$, we define $\bar{\mathsf{MSC}}(\bar{e})$ as follows:*

$$\bar{\mathsf{MSC}}(\lambda x.e) = \lambda x.\mathsf{MSC}(e)$$
$$\bar{\mathsf{MSC}}(\bar{e}) = \bar{e} \qquad \textit{for any } \bar{e} \textit{ that is not a lambda}$$

Property I.35. *For any atomic source expression $\bar{e}$, $\mathtt{bind}\,\mathtt{x} = \bar{\mathsf{MSC}}(\bar{e})\,\mathtt{in}\,\mathtt{x}$ is a valid MSC-form.*

Proof. The extrusion property is ensured by the conditions on lambda-expressions in the definition I.33. □

Definition I.36 (Binding context). *We call binding context (noted C) a canonical form ending with a hole:*

$$\textbf{\textit{Binding context}} \quad C \quad ::= \quad [\,] \mid \mathtt{bind}\,\mathsf{x} = a\ \mathtt{in}\ \kappa \tag{47}$$

and we use the usual notation $C[\kappa]$ for denoting the canonical form obtained by replacing the hole in C by κ.

When speaking of the set of (sub)expressions defined by a binding context C, it will denote the set of the expressions $\lceil C[\mathsf{x}]\rceil$ for any binding variable x in the scope of the hole in C.

Definition I.37. *We define the operator $\lceil e\rceil_C$ as follows:*

$$\lceil e\rceil_{[\,]} = e$$

$$\lceil e\rceil_{(\mathtt{bind}\,\mathsf{x}=a\ \mathtt{in}\ C)} = (\lceil(e)\rceil_C)\{\lceil a\rceil/\mathsf{x}\}$$

Definition I.38. *We define the operator $e \setminus C$ as follows:*

$$e \setminus [\,] = e$$

$$e \setminus (\mathtt{bind}\,\mathsf{x}=a\ \mathtt{in}\ C) = (e\{\mathsf{x}/\lceil a\rceil\}) \setminus C$$

Property I.39. *For any binding context C and expression e, we have $\lceil e \setminus C\rceil_C \equiv_\alpha \lceil e\rceil_C$ and $(\lceil e\rceil_C) \setminus C \equiv_\alpha e \setminus C$.*

Proof. Straightforward. □

Property I.40 (Decomposition of form derivations). *If $\Gamma \vdash e : t$ is a form derivation with the root being a $[\vee]$ rule doing the substitution $e'\{e_\mathsf{x}/\mathsf{x}\}$, then there exists a binding context C such that $\mathsf{MSC}(e) \equiv_\alpha C[\mathtt{bind}\,\mathsf{x} = \bar{\mathsf{MSC}}(e_\mathsf{x} \setminus C)\ \mathtt{in}\ \mathsf{MSC}(e' \setminus C)\,]$.*

Proof. First, we can easily deduce from the fact that our derivation $\Gamma \vdash e'\{e_\mathsf{x}/\mathsf{x}\} : t$ is normalized (as it is a form derivation) that the premise $\Gamma \vdash e_\mathsf{x} : s$ is an atom derivation.

We know that e_x appears in e (as e' contains x, see I.7). Thus, we can deduce that $\mathsf{MSC}(e)$ contains a definition for an atom a that unwinds to e_x. More formally, we know that there exists a binding context C, an atom a and a canonical form κ such that $\mathsf{MSC}(e) \equiv_\alpha C[\mathtt{bind}\,\mathsf{x} = a\ \mathtt{in}\ \kappa\,]$ with $\lceil\lceil a\rceil\rceil_C \equiv_\alpha e_\mathsf{x}$.

The expression e_x could contain some subexpressions that are not binding variables and that have no occurrence of a lambda variable defined in e_x. Thus, these subexpressions must be defined through atomic bindings in C (the unwindings of the corresponding definitions are necessarily smaller than e_x by $\leq$ as they are subexpressions of e_x). The expression e' could also contain some such subexpressions. The ones whose unwnding is smaller than e_x according to $\leq$ must be defined through atomic bindings in C too. No other expression should be defined in C (and each of these subexpressions must be associated to a unique binding variable) or it would contradict the properties of MSC-forms.

Under the context C, the expression $e_\mathsf{x} \setminus C$ unwinds to e_x (I.39). Moreover, as $\Gamma \vdash e_\mathsf{x} : s$ is an atom derivation, $e_\mathsf{x} \setminus C$ must be an atomic source expression. Thus, we can deduce from I.35 that $\bar{\mathsf{MSC}}(e_\mathsf{x} \setminus C)$ can be used in place of the atom a without breaking any property of the MSC-form, and thus by unicity of the MSC-form we can conclude that $a \equiv_\alpha \bar{\mathsf{MSC}}(e_\mathsf{x} \setminus C)$.

Similarly, the expression $e' \setminus (C[\mathtt{bind}\,\mathsf{x} = \bar{\mathsf{MSC}}(e_\mathsf{x} \setminus C)\ \mathtt{in}\ [\,]\,])$ unwinds to $e'\{e_\mathsf{x}/\mathsf{x}\}$ under the context $C[\mathtt{bind}\,\mathsf{x} = \bar{\mathsf{MSC}}(e_\mathsf{x} \setminus C)\ \mathtt{in}\ [\,]\,]$ (I.39). As the derivation is normalized, e_x cannot be a subexpression of e' (I.7), thus $e' \setminus C \equiv_\alpha e' \setminus (C[\mathtt{bind}\,\mathsf{x} = \bar{\mathsf{MSC}}(e_\mathsf{x} \setminus C)\ \mathtt{in}\ [\,]\,])$ and thus $e' \setminus C$ also unwinds to $e'\{e_\mathsf{x}/\mathsf{x}\}$. Thus, $\mathsf{MSC}(e' \setminus C)$ can be used in place of κ without breaking any property of the MSC-form (note that it only contains top-level bindings for expressions greater than e_x by $\leq$, as the smaller ones have been put in C). By unicity of the MSC-form, we conclude that $\kappa \equiv_\alpha \mathsf{MSC}(e' \setminus C)$, and thus $\mathsf{MSC}(e) \equiv_\alpha C[\mathtt{bind}\,\mathsf{x} = \bar{\mathsf{MSC}}(e_\mathsf{x} \setminus C)\ \mathtt{in}\ \mathsf{MSC}(e' \setminus C)\,]$. □

Theorem I.41 (Completeness).
If $\Gamma \vdash e : t$ is a form derivation, then $\exists \Bbbk, t'.\ \Gamma \vdash_{\mathcal{A}} [\mathsf{MSC}(e) \mid \Bbbk] : t'$ with $t' \leq_P t$.
If $\Gamma \vdash \bar{e} : t$ is an atom derivation (with $\bar{e}$ an atomic source expression), then $\exists \mathbb{a}, t'.\ \Gamma \vdash_{\mathcal{A}} [\bar{\mathsf{MSC}}(\bar{e}) \mid \mathbb{a}] : t'$ with $t' \leq_P t$.

Proof. We proceed by induction on the depth of $\Gamma \vdash e : t$.

We consider the root of the derivation (the cases up to $[\in_2]$ are for atom derivations, the cases after are for form derivations):

[Const] Trivial.

[Ax$_\lambda$] Trivial.

[→I] We have $\bar{e} \equiv \lambda x.\ e$ and thus $\bar{\mathsf{MSC}}(\bar{e}) \equiv_\alpha \lambda x.\ \mathsf{MSC}(e)$.
The premise of a normalised [→I] rule must be a form derivation (see I.7). Thus, by induction on this premise, we get $\Gamma, x : \mathbf{u} \vdash_{\mathcal{A}} [\mathsf{MSC}(e) \mid \Bbbk] : t'$ (with $t' \leq_P t$). We can thus derive $\Gamma \vdash_{\mathcal{A}} [\lambda x.\ \mathbb{MSC}(e) \mid \lambda(\mathbf{u}, \Bbbk)] : \mathbf{u} \to t'$ and we have $\mathbf{u} \to t' \leq_P \mathbf{u} \to t$, which concludes this case.

[→E] We have $\bar{e} \equiv x_1 x_2$ and thus $\bar{\mathsf{MSC}}(\bar{e}) \equiv_\alpha x_1 x_2$.
As our derivation is normalised, we know that the second premise, $\Gamma \vdash x_2 : t_1$, is a [Inst∧≤] pattern whose premise is a [Ax$_\vee$], and with no [≤] rule. Thus, we know that there exists Σ_2 such that $\Gamma(x_2)\Sigma_2 \simeq t_1$. Similarly, the first premise, $\Gamma \vdash x_1 : t_1 \to t_2$, is also a [Inst∧≤] pattern whose premise is a [Ax$_\vee$] (with possibly a [≤] rule). Thus, we know that there exists Σ_1 such that $\Gamma(x_1)\Sigma_1 \leq t_1 \to t_2$.
Consequently, and by definition of $\circ$, we know that $(\Gamma(x_1)\Sigma_1) \circ (\Gamma(x_2)\Sigma_2) \leq t_2$. We can thus derive $\Gamma \vdash_{\mathcal{A}} [x_1 x_2 \mid @(\Sigma_1, \Sigma_2)] : t'$ (with $t' \simeq (\Gamma(x_1)\Sigma_1) \circ (\Gamma(x_2)\Sigma_2)$) such that $t' \leq t_2$, which concludes this case.

[×I] We have $\bar{e} \equiv (x_1, x_2)$ and thus $\bar{\mathsf{MSC}}(\bar{e}) \equiv_\alpha (x_1, x_2)$.
As our derivation is normalized, both premises can only be a [Ax$_\vee$]. Thus, we can deduce that there exists two renamings of polymorphic variables ρ_1 and ρ_2 such that $\Gamma(x_1)\rho_1 \simeq t_1$ and $\Gamma(x_2)\rho_2 \simeq t_2$. Thus, we can derive $\Gamma \vdash_{\mathcal{A}} [(x_1, x_2) \mid (\rho_1, \rho_2)] : t_1 \times t_2$.

[×E$_1$] We have $\bar{e} \equiv \pi_1 x$ and thus $\bar{\mathsf{MSC}}(\bar{e}) \equiv_\alpha \pi_1 x$.
As our derivation is normalised, we know that the premise, $\Gamma \vdash x : t_1 \times t_2$, is a [Inst∧≤] pattern whose premise is a [Ax$_\vee$]. Thus, we know that there exists Σ such that $\Gamma(x)\Sigma \leq t_1 \times t_2$. Consequently, and by definition of $\boldsymbol{\pi}_1$, we know that $\boldsymbol{\pi}_1(\Gamma(x)\Sigma) \leq t_1$. We can thus derive $\Gamma \vdash_{\mathcal{A}} [\pi_1 x \mid \pi(\Sigma)] : t'$ (with $t' \simeq \boldsymbol{\pi}_1(\Gamma(x)\Sigma)$) such that $t' \leq t_1$, which concludes this case.

[×E$_2$] Similar to the previous case.

[$\mathbb{0}$] We have $\bar{e} \equiv (x{\in}\tau)\,?\,x_1 : x_2$ and thus $\bar{\mathsf{MSC}}(\bar{e}) \equiv_\alpha (x{\in}\tau)\,?\,x_1 : x_2$.
As our derivation is normalised, we know that the premise, $\Gamma \vdash x : \mathbb{0}$, is a [Inst∧≤] pattern whose premise is a [Ax$_\vee$], and with no [≤] rule. Thus, we know that there exists Σ such that $\Gamma(x)\Sigma \simeq \mathbb{0}$.
We can thus derive $\Gamma \vdash_{\mathcal{A}} [(x{\in}\tau)\,?\,x_1 : x_2 \mid \mathbb{0}(\Sigma)] : \mathbb{0}$.

[$\in_1$] We have $\bar{e} \equiv (x{\in}\tau)\,?\,x_1 : x_2$ and thus $\bar{\mathsf{MSC}}(\bar{e}) \equiv_\alpha (x{\in}\tau)\,?\,x_1 : x_2$.
As our derivation is normalised, we know that the first premise, $\Gamma \vdash x : \tau$, is a [Inst∧≤] pattern whose premise is a [Ax$_\vee$]. Thus, we know that there exists Σ such that $\Gamma(x)\Sigma \leq \tau$. Similarly, the second premise, $\Gamma \vdash x_1 : t_1$, can only be a [Ax$_\vee$]. Thus, we know that there exists a renaming of polymorphic variables ρ such that $\Gamma(x_1)\rho \simeq t_1$.
We can thus derive $\Gamma \vdash_{\mathcal{A}} [(x{\in}\tau)\,?\,x_1 : x_2 \mid \in_1(\Sigma)] : \Gamma(x_1)$ with $\Gamma(x_1) \leq_P t_1$.

[$\in_2$] Similar to the previous case.

[Ax$_\vee$] Trivial.

[≤] Straightforward induction.

[Inst] Straightforward induction.

[$\wedge$] By induction on the premises, we get $\forall i \in I.\ \Gamma \vdash_{\mathcal{A}} [\mathsf{MSC}(e) \mid \mathbb{k}_i] : t'_i$ with $t'_i \leq_{\mathrm{P}} t_i$. Thus, we can derive $\Gamma \vdash_{\mathcal{A}} [\mathsf{MSC}(e) \mid \bigwedge(\{\mathbb{k}_i\}_{i\in I})] : \bigwedge_{i\in I} t'_i$ (with $\bigwedge_{i\in I} t'_i \leq_{\mathrm{P}} \bigwedge_{i\in I} t_i$).

[$\vee$] By using I.40, we know that there exists C such that $\mathsf{MSC}(e) \equiv_\alpha C[\mathtt{bind}\,\mathsf{x} = \mathsf{M\bar{S}C}(e_{\mathsf{x}} \setminus C)\ \mathtt{in}\ \mathsf{MSC}(e' \setminus C)\,]$ (with $e_{\mathsf{x}} \setminus C$ being an atomic source expression).
The unwiding of the definitions in C are necessarily smaller than e_{x}. Thus, none of them can be defined by a [$\vee$] rule in the current derivation for $\Gamma \vdash e'\{e_{\mathsf{x}}/\mathsf{x}\} : t$. As a structural rule (or a [$\to$I$\wedge$] pattern) can only appear as the first premise of a [$\vee$] rule (I.11), none of the expressions defined in C are typed in the current derivation. Thus, we can easily replace them, in the current derivation, by the associated binding variable in C. It gives us a derivation for $\Gamma \vdash (e'\{e_{\mathsf{x}}/\mathsf{x}\}) \setminus C : t$, or written differently, for $\Gamma \vdash (e' \setminus C)\{(e_{\mathsf{x}} \setminus C)/\mathsf{x}\} : t$.
By induction on the premises of this new derivation, we get $\Gamma \vdash_{\mathcal{A}} [\mathbb{M}\bar{\mathbb{S}}\mathbb{C}(e_{\mathsf{x}} \setminus C) \mid \mathbb{a}] : s'$ (with $s' \leq_{\mathrm{P}} s$) and $\forall i \in I.\ \Gamma, \mathsf{x} : s \wedge \mathbf{u}_i \vdash_{\mathcal{A}} [\mathsf{MSC}(e' \setminus C) \mid \mathbb{k}_i] : t_i$ (with $t_i \leq_{\mathrm{P}} t$). By monotonicity (I.30), we can derive $\forall i \in I.\ \Gamma, \mathsf{x} : s' \wedge \mathbf{u}_i \vdash_{\mathcal{A}} [\mathsf{MSC}(e' \setminus C) \mid \mathbb{k}'_i] : t'_i$ (with $t'_i \leq_{\mathrm{P}} t_i \leq_{\mathrm{P}} t$). We can thus derive $\Gamma \vdash_{\mathcal{A}} [\mathtt{bind}\,\mathsf{x} = \mathsf{M\bar{S}C}(e_{\mathsf{x}} \setminus C)\ \mathtt{in}\ \mathsf{MSC}(e' \setminus C) \mid \mathbb{k}] : \bigvee_{i\in I} t'_i$ with $\mathbb{k} = \mathtt{keep}\ (\mathbb{a}, \{(\mathbf{u}_i, \mathbb{k}'_i)\}_{i\in I})$.
From that, we can easily derive $\Gamma \vdash_{\mathcal{A}} [\mathsf{MSC}(e) \mid \mathbb{k}'] : \bigvee_{i\in I} t'_i$ with $\mathbb{k}'$ obtained by adding to the root of $\mathbb{k}$ a $\mathtt{skip}$ annotation for each definition in C.

□

Property I.42. *Any derivation $\Gamma \vdash e : t$ can be transformed into a form derivation.*

Proof. We first normalize $\Gamma \vdash e : t$ using I.11. If the derivation we obtain is not a form derivation, that is, if it contains a structural rule and its root is not a [Inst$\wedge\leq$] pattern with a [$\vee$] rule as premise, then we replace its root D (or, if its root is a [Inst$\wedge\leq$] pattern, the premise of this [Inst$\wedge\leq$] pattern) with:

$$[\vee]\ \dfrac{\dfrac{D}{\Gamma \vdash e : t} \qquad [\mathrm{Ax}_\vee]\ \dfrac{}{\Gamma, \mathsf{x} : t \vdash \mathsf{x} : t}}{\Gamma \vdash \mathsf{x}\{e/\mathsf{x}\} : t}$$

It is straightforward to check that the derivation we obtain is a form derivation. □

Corollary I.43 (Completeness). *If $\Gamma \vdash e : t$, then $\exists \mathbb{k}, t'.\ \Gamma \vdash_{\mathcal{A}} [\mathsf{MSC}(e) \mid \mathbb{k}] : t'$ with $t' \leq_P t$.*

Proof. Direct application of I.41 after using I.42 on $\Gamma \vdash e : t$. □

I.3 Annotations reconstruction system

In the following proofs, we assume that all the binding variables used in canonical forms are fresh (there is no conflict with another binding in the canonical form, nor with a binding variable already in the environment).

I.3.1 Termination.

Definition I.44. *We define the successors of a result $\mathbb{R}$ as the set of pairs $\{(\Gamma_i, \mathcal{H}_i)\}_{i\in I}$ defined as follows:*

$$\begin{aligned}\texttt{succ}(\texttt{Ok}(\mathcal{H})) &\overset{\text{def}}{=} \{\}\\ \texttt{succ}(\texttt{Fail}) &\overset{\text{def}}{=} \{\}\\ \texttt{succ}(\texttt{Split}(\Gamma, \mathcal{H}_1, \mathcal{H}_2)) &\overset{\text{def}}{=} \{(\Gamma, \mathcal{H}_1)\} \cup \{(\{(\mathtt{x} : \neg\boldsymbol{u})\}, \mathcal{H}_2) \mid (\mathtt{x} : \boldsymbol{u}) \in \Gamma\}\\ \texttt{succ}(\texttt{Subst}(\{\psi_i\}_{i\in I}, \mathcal{H}_1, \mathcal{H}_2)) &\overset{\text{def}}{=} \{(\varnothing, \mathcal{H}_1), (\varnothing, \mathcal{H}_2)\}\\ \texttt{succ}(\texttt{Var}\ (\mathtt{x}, \mathcal{H}_1, \mathcal{H}_2)) &\overset{\text{def}}{=} \{(\{(\mathtt{x} : \mathbb{1})\}, \mathcal{H}_1), (\varnothing, \mathcal{H}_2)\}\end{aligned}$$

Definition I.45. *Let η a canonical form or atom. Let Γ_1, Γ_2 two environments and $\mathcal{H}_1, \mathcal{H}_2$ two annotations.*

We define the notion of derivation step for $\vdash_{\mathcal{R}}$ and η from the pair $(\Gamma_1, \mathcal{H}_1)$ to the pair $(\Gamma_2, \mathcal{H}_2)$ as follows: $(\Gamma_1, \mathcal{H}_1) \rightsquigarrow_{\vdash_{\mathcal{R}},\eta} (\Gamma_2, \mathcal{H}_2) \Leftrightarrow \exists\psi, \psi'.\ \exists\Gamma \leq \Gamma_1\psi.\ \Gamma \vdash_{\mathcal{R}} \langle \eta \mid \mathcal{H}_1\psi' \rangle \Rightarrow \mathbb{R}, \Gamma_2 \leq \Gamma_1\psi$ and $(\Gamma_2', \mathcal{H}_2) \in \texttt{succ}(\mathbb{R})$ for some $\Gamma_2' \geq \Gamma_2$.

We also define the notion of derivation step for $\vdash^*_{\mathcal{R}}$ and η from the pair $(\Gamma_1, \mathcal{H}_1)$ to the pair $(\Gamma_2, \mathcal{H}_2)$ as follows: $(\Gamma_1, \mathcal{H}_1) \rightsquigarrow_{\vdash^*_{\mathcal{R}},\eta} (\Gamma_2, \mathcal{H}_2) \Leftrightarrow \exists\psi, \psi'.\ \exists\Gamma \leq \Gamma_1\psi.\ \Gamma \vdash^*_{\mathcal{R}} \langle \eta \mid \mathcal{H}_1\psi' \rangle \Rightarrow \mathbb{R}, \Gamma_2 \leq \Gamma_1\psi$ and $(\Gamma_2', \mathcal{H}_2) \in \texttt{succ}(\mathbb{R})$ for some $\Gamma_2' \geq \Gamma_2$.

We use the notation $(\Gamma, \mathcal{H}) \rightsquigarrow^*_{\vdash_{\mathcal{R}},\eta} (\Gamma', \mathcal{H}')$ for denoting a finite sequence of $\rightsquigarrow_{\vdash_{\mathcal{R}},\eta}$ steps starting with $(\Gamma, \mathcal{H})$ and ending with $(\Gamma', \mathcal{H}')$. We also use the notation $(\Gamma, \mathcal{H}) \rightsquigarrow^\infty_{\vdash_{\mathcal{R}},\eta}$ to denote the existence of a sequence of $\rightsquigarrow_{\vdash_{\mathcal{R}},\eta}$ steps starting with $(\Gamma, \mathcal{H})$ that can be prolonged infinitely.

Lemma I.46. *Let $\Gamma, \mathcal{H}, \Gamma'$ and $\mathcal{H}'$ such that $\exists\psi.\ \Gamma \leq \Gamma'\psi$ and $\exists\psi.\ \mathcal{H} \equiv \mathcal{H}'\psi$. If $(\Gamma, \mathcal{H}) \rightsquigarrow^\infty_{\vdash_{\mathcal{R}},\eta}$, then $(\Gamma', \mathcal{H}') \rightsquigarrow^\infty_{\vdash_{\mathcal{R}},\eta}$. If $(\Gamma, \mathcal{H}) \rightsquigarrow^\infty_{\vdash^*_{\mathcal{R}},\eta}$, then $(\Gamma', \mathcal{H}') \rightsquigarrow^\infty_{\vdash^*_{\mathcal{R}},\eta}$.*

Proof. Straightfoward consequence of the definitions of $\rightsquigarrow_{\vdash_{\mathcal{R}},\eta}$ and $\rightsquigarrow_{\vdash^*_{\mathcal{R}},\eta}$. The first element of the chain can be replaced by $(\Gamma', \mathcal{H}')$ without having to change the next elements nor the derivations justifying each step. □

Lemma I.47. *If $(\Gamma, \bigwedge(\{\mathcal{H}_i\}_{i\in I}, S)) \rightsquigarrow^\infty_{\vdash_{\mathcal{R}},\eta}$, then $\exists i \in I.\ (\Gamma, \mathcal{H}_i) \rightsquigarrow^\infty_{\vdash^*_{\mathcal{R}},\eta}$.*

Proof. We know that in the chain $(\Gamma, \bigwedge(\{\mathcal{H}_i\}_{i\in I}, S)) \rightsquigarrow^\infty_{\vdash_{\mathcal{R}},\eta}$, all the derivations have as root [Inter₁], [Inter₂] or [Inter₃] (because the result of [InterOk] and [InterEmpty] cannot have any successor). Thus, we know that there exists $i \in I$ such that we can find in our chain arbitrarily many derivations applying the rule [Inter₃] on an annotation issued from the same initial $\mathcal{H}_i$. Let's extract from our chain the sub-sequence associated to these derivations, and replace each of its pairs $(\Gamma', \bigwedge(\{\mathcal{H}'\} \cup S_1, S_2))$ (with $\mathcal{H}'$ the annotation issued from $\mathcal{H}_i$) with the pair $(\Gamma', \mathcal{H}')$.

The sequence we obtain must start with $(\Gamma', \mathcal{H}_i')$ for some Γ' and $\mathcal{H}_i'$ such that $\exists\psi.\ \Gamma' \leq \Gamma\psi$ and $\exists\psi.\ \mathcal{H}_i' \equiv \mathcal{H}_i\psi$. Moreover, two consecutive pairs $(\Gamma_i', \mathcal{H}_i')$ and $(\Gamma_{i+1}', \mathcal{H}_{i+1}')$ satisfy $(\Gamma_i', \mathcal{H}_i') \rightsquigarrow_{\vdash^*_{\mathcal{R}},\eta} (\Gamma_{i+1}', \mathcal{H}_{i+1}')$: it can be deduced from the premise of the associated [Inter₃] rule. We thus have $(\Gamma', \mathcal{H}_i') \rightsquigarrow^\infty_{\vdash^*_{\mathcal{R}},\eta}$ and thus by I.46 $(\Gamma, \mathcal{H}_i) \rightsquigarrow^\infty_{\vdash^*_{\mathcal{R}},\eta}$. □

Lemma I.48. *If $(\Gamma, \mathcal{H}) \rightsquigarrow^\infty_{\vdash^*_{\mathcal{R}},\eta}$, then $(\Gamma, \mathcal{H}) \rightsquigarrow^\infty_{\vdash_{\mathcal{R}},\eta}$.*

Proof. Let's assume we have $(\Gamma, \mathcal{H}) \rightsquigarrow^\infty_{\vdash^*_{\mathcal{R}},\eta}$ and let's build a chain $(\Gamma, \mathcal{H}) \rightsquigarrow^\infty_{\vdash_{\mathcal{R}},\eta}$. For that, we define a process that transforms an infinite chain, initially $(\Gamma, \mathcal{H}) \rightsquigarrow^\infty_{\vdash^*_{\mathcal{R}},\eta}$, into some steps $(\Gamma, \mathcal{H}) \rightsquigarrow^*_{\vdash_{\mathcal{R}},\eta} (\Gamma', \mathcal{H}')$ (of length ≥ 1) and a new chain $(\Gamma', \mathcal{H}') \rightsquigarrow^\infty_{\vdash_{\mathcal{R}},\eta}$. This process can be used iteratively to arbitrarily extend the $\rightsquigarrow_{\vdash_{\mathcal{R}},\eta}$ chain.

We consider the derivation associated with the first step $(\Gamma, \mathcal{H}) \rightsquigarrow_{\vdash^*_{\mathcal{R}},\eta} (\Gamma'', \mathcal{H}'')$ of the input chain, by supposing without loss of generality that the associated derivation has a judgement

$\Gamma \vdash_{\mathcal{R}} \langle \eta \mid \mathcal{H} \rangle \Rightarrow \mathbb{R}$ for some $\mathbb{R}$ (we apply I.46 if it is not the case). We proceed by induction on this derivation:

- If the root is [Stop], the premise gives $(\Gamma, \mathcal{H}) \rightsquigarrow_{\vdash_{\mathcal{R}}, \eta} (\Gamma'', \mathcal{H}'')$, and we return this together with the tail $(\Gamma'', \mathcal{H}'') \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \eta}$ of the input chain.
- If the root is [Iterate$_1$], the first premise gives $(\Gamma, \mathcal{H}) \rightsquigarrow_{\vdash_{\mathcal{R}}, \eta} (\Gamma, \mathcal{H}_1)$, and by induction on the second premise we get a chain $(\Gamma, \mathcal{H}_1) \rightsquigarrow^{*}_{\vdash_{\mathcal{R}}, \eta} (\Gamma', \mathcal{H}')$ and $(\Gamma', \mathcal{H}') \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \eta}$ for some Γ' and $\mathcal{H}'$.
- If the root is [Iterate$_2$], the first premise gives $(\Gamma, \mathcal{H}) \rightsquigarrow_{\vdash_{\mathcal{R}}, \eta} (\Gamma, \mathcal{H}_1)$ and $(\Gamma, \mathcal{H}) \rightsquigarrow_{\vdash_{\mathcal{R}}, \eta} (\Gamma, \mathcal{H}_2)$. Let $\mathcal{H}' = \bigwedge(\mathcal{H}_1 \psi_i, \varnothing)$.
  By induction on the last premise, we get a chain $(\Gamma, \mathcal{H}') \rightsquigarrow^{*}_{\vdash_{\mathcal{R}}, \eta} (\Gamma''', \mathcal{H}''')$ for some Γ''' and $\mathcal{H}'''$ and a chain $(\Gamma''', \mathcal{H}''') \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, \eta}$. By concatenating these two chains and applying I.47 on the resulting chain, we know there exists a chain $(\Gamma, \mathcal{H}_1 \psi_i) \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \eta}$ for some $i \in I$, or a chain $(\Gamma, \mathcal{H}_2) \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \eta}$.
  In the first case, we apply I.46 to get a chain $(\Gamma, \mathcal{H}_1) \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \eta}$ (we have $\psi_i \# \Gamma$), and we return it with the chain $(\Gamma, \mathcal{H}) \rightsquigarrow_{\vdash_{\mathcal{R}}, \eta} (\Gamma, \mathcal{H}_1)$. In the other case, we directy return $(\Gamma, \mathcal{H}_2) \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \eta}$ with $(\Gamma, \mathcal{H}) \rightsquigarrow_{\vdash_{\mathcal{R}}, \eta} (\Gamma, \mathcal{H}_2)$.

□

Lemma I.49. *If* $(\Gamma, \bigwedge(\{\mathcal{H}_i\}_{i \in I}, S)) \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, \eta}$, *then* $\exists i \in I.\ (\Gamma, \mathcal{H}_i) \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, \eta}$.

Proof. Combination of I.47 and I.48. □

Lemma I.50. *For any* η, Γ *and* $\mathcal{H}$, *there is no chain* $(\Gamma, \mathcal{H}) \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, \eta}$.

Proof. We proceed by structural induction on η.

Let $(\Gamma, \mathcal{H}) \rightsquigarrow_{\vdash_{\mathcal{R}}, \eta} (\Gamma', \mathcal{H}')$ be the first step of a chain. We show, with another induction on the depth of the derivation associated with this first step, that the chain cannot be infinitely extended.

Let's first consider the case of an atom ($\eta \equiv a$ and $\mathcal{H} \equiv \mathcal{A}$).

If $\mathcal{A}$ is `typ` or `untyp`, only the rules [Ok] or [Fail] can apply, whose result has no successor.

If $\mathcal{A}$ is an intersection annotation $\bigwedge(\{\mathcal{A}_i\}_{i \in I}, S)$, we get by induction that for any $i \in I$ there is no chain $(\Gamma, \mathcal{A}_i) \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, a}$. By applying (the contrapositive of) I.49, we deduce that there is no chain $(\Gamma, \bigwedge(\{\mathcal{A}_i\}_{i \in I}, S)) \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, a}$.

If $\mathcal{A}$ is an annotation $\lambda(\mathbf{u}, \mathcal{K})$, then only [LambdaEmpty] or [Lambda] can apply for the first step. By the absurd, let's suppose we have a chain $(\Gamma, \lambda(\mathbf{u}, \mathcal{K})) \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, a}$. As the result of [LambdaEmpty] has no possible successor, the derivation of the first step of this chain must use a [Lambda] rule as root, and so must the derivations of the next steps. From the premises of these derivations, we can construct a chain $(\Gamma, \mathcal{K}) \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, a}$, and by applying I.48 we obtain a chain for $(\Gamma, \mathcal{K}) \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, a}$, which contradicts the induction hypotheses.

Otherwise, $\mathcal{A}$ is an annotation `infer`. Depending on a:

c Trivial (only [Const] applies)

x Trivial (only [AxOk] and [AxFail] apply)

$(\mathbf{x}_1, \mathbf{x}_2)$ There can be at most two derivations using [PairVar] in the chain, because for any of them, either its successor uses the annotation `untyp` or it is still using the annotation `infer` but with $\mathrm{x}_i \in \mathrm{dom}(\Gamma)$. The only remaining rule that can be applied is [PairOk], and it has no successor.

$\pi_i \mathbf{x}$ There can be at most one derivation using [ProjVar] in the chain, because either its successor uses the annotation `untyp` or it is still using the annotation `infer` but with $\mathrm{x} \in \mathrm{dom}(\Gamma)$. Moreover, all the possible successors for [ProjInfer] use the annotation `typ`.

$\mathtt{x}_1\mathtt{x}_2$ Similar to the previous case.

$(\mathtt{x}{\in}\tau)\,?\,\mathtt{x}_1 : \mathtt{x}_2$ Similar to the previous case, using for the [CaseSplit] rule the guarantee that its successors will either have $\Gamma(\mathrm{x}) \le \tau$ or $\Gamma(\mathrm{x}) \le \neg\tau$.

$\lambda x.\kappa$ The only rule applicable in this case is [LambdaInfer], and we can easily conclude by induction on its premise.

Now, let's consider the case of a canonical form ($\eta \equiv \kappa$ and $\mathcal{H} \equiv \mathcal{K}$).

If $\mathcal{K}$ is an intersection annotation, then we conclude in a similar way as for the intersection case of the atoms.

Otherwise, and if κ is a variable x, then the annotation can only be `typ`, `untyp` or `infer`. Only the rules [Ok], [Fail], [FormVar] or [FormOk] can apply, and we conclude trivially in each case.

Lastly, if κ is a binding, let's say $\mathtt{bind}\,\mathrm{x} = a\ \mathtt{in}\ \kappa'$, the annotation $\mathcal{K}$ can be one of those (the case for an intersection annotation has already been treated):

`infer` We can conclude by using the induction hypotheses.

`try-skip` $(\mathcal{K}_1)$ We cannot have arbitrarily many derivations applying [BindTrySkip$_3$] on κ in the chain, because it would give, for some Γ' and $\mathcal{K}'$, a chain $(\Gamma', \mathcal{K}') \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \kappa'}$, and thus $(\Gamma', \mathcal{K}') \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, \kappa'}$ (I.48), which would contradict the induction hypotheses.
Thus, extending the chain infinitely would necessarily require to apply [BindTrySkip$_1$] or [BindTrySkip$_2$] at some point, allowing us to conclude with the next cases.

`try-keep` $(\mathcal{A}, \mathcal{K}_1, \mathcal{K}_2)$ This case is similar to the previous one.

`skip` $(\mathcal{K}_1)$ We cannot have arbitrarily many derivations applying [BindSkip$_1$] or [BindSkip$_2$] on κ in the chain, because it would give, for some Γ' and $\mathcal{K}'$, a chain $(\Gamma', \mathcal{K}') \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \kappa'}$, and thus $(\Gamma', \mathcal{K}') \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, \kappa'}$ (I.48), which would contradict the induction hypotheses.

`propagate` $(\mathcal{A}, \mathbb{\Gamma}, \mathcal{S}, \mathcal{S}')$ There cannot be arbitrarily many consecutive derivations in the chain that can use [BindProp$_1$] on κ. Thus, extending the chain infinitely would necessarily require to apply [BindProp$_2$] at some point, allowing us to conclude with the next case.

`keep` $(\mathcal{A}, \mathcal{S}, \mathcal{S}')$ Let's suppose we have $(\Gamma, \mathtt{keep}\ (\mathcal{A}, \mathcal{S}, \mathcal{S}')) \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, \kappa}$.
As there cannot be arbitrarily many consecutive derivations in the chain that can use [BindProp$_1$] or [BindProp$_2$], we know that the chain features arbitrarily many [BindSplit$_1$], [BindSplit$_2$] or [BindSplit$_3$] applied on κ. Also, arbitrarily many of them must apply to an annotation that is issued from the same initial annotation $\mathcal{K}'$ $((\_, \mathcal{K}') \in \mathcal{S})$ (by considering that when a split occurs, both new annotations are issued from the annotation that was splitted). We can thus extract from the chain the sub-sequence corresponding to the derivations that apply one of the [BindSplit] rules on an annotation issued from $\mathcal{K}'$. From the premises of the derivations of this sub-sequence, we can extract a chain $(\Gamma', \mathcal{K}') \rightsquigarrow^{\infty}_{\vdash^*_{\mathcal{R}}, \kappa'}$ for some Γ', and thus using I.48 a chain $(\Gamma', \mathcal{K}') \rightsquigarrow^{\infty}_{\vdash_{\mathcal{R}}, \kappa'}$, which contradicts the induction hypotheses.

□

Theorem I.51 (Termination). *For any Γ, κ and $\mathcal{K}$, applying the reconstruction deduction rules on the input $\Gamma \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle$ always terminate: either it gets stuck or it derives a judgement $\Gamma \vdash^*_{\mathcal{R}} \langle \kappa \mid \mathcal{K} \rangle \Rightarrow \mathbb{R}$ for some $\mathbb{R}$.*

Proof. All the recursive premises in the $\vdash_{\mathcal{R}}$ rules are either applied on a strict subexpression, or on the same expression but with a strictly decreasing annotation for some straightforward partial order. For the $\vdash^*_{\mathcal{R}}$ rules, we justify termination using lemma I.50. □