Liquid Intersection Types

Mário Pereira Sandra Alves Mário Florido

Technical Report Series: DCC-2014-04

Departamento de Ciência de Computadores
&
Laboratório de Inteligência Artificial e de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL

Tel: 220 402 900 Fax: 220 402 950
http://www.dcc.fc.up.pt/Pubs/

Liquid Intersection Types

Mário Pereira Sandra Alves Mário Florido
University of Porto, Department of Computer Science & LIACC
{mariopereira,sandra,amf}@dcc.fc.up.pt

April 19, 2014

Abstract

We present a new type system combining refinement types and the expressiveness of intersection type discipline. The use of such features makes it possible to derive more precise types than in the original refinement system. We have been able to prove several interesting properties for our system (including subject reduction) and developed an inference algorithm, which we proved to be sound.

1 Introduction

Refinement types [9] state complex program invariants, by augmenting type systems with logical predicates. A refinement type of the form $\{\nu : B \mid \phi\}$ stands for the set of values from basic type B restricted to the filtering predicate (refinement) ϕ. A subtyping relation exists for refinement types, which will generate implication conditions :

$$\frac{\Gamma; \nu : B \vdash \phi \Rightarrow \psi}{\Gamma \vdash \{\nu : B \mid \phi\} <: \{\nu : B \mid \psi\}}$$

One idea behind the use of such type systems is to perform type-checking using SMTs (Satisfability Modulo Theories) [13], discharging conditions as the above $\phi \Rightarrow \psi$. However, the use of arbitrary boolean terms as refinement expressions leads to undecidable type systems, both for type checking and inference.

Liquid Types [12, 14] present a system capable of automatically infer refinement types, by means of two main restrictions to a general refinement type system: every refinement predicate is a conjunction of expressions exclusively taken from a global, user-supplied set (denoted $\mathbb{Q}$) of logical qualifiers (simple predicates over program variables, the value variable ν and the variable placeholder $\star$); and a conservative (hence decidable) notion of subtyping.

Despite the interest of Liquid Types, some situations arise where the inference procedure infers poorly accurate types. For example, considering $\mathbb{Q} = \{\nu \geq 0, \nu \leq 0\}$ and the term $neg \equiv \lambda x. -x$, Liquid Types infer for *neg* the type $x : \{0 \leq \nu \wedge 0 \geq \nu\} \rightarrow \{0 \leq \nu \wedge 0 \geq \nu\}$ (throughout this paper we write $\{\phi\}$ instead of $\{\nu : B \mid \phi\}$ whenever B is clear from the context). This type cannot be taken as a precise description of the *neg* function's behavior, since it is not expressed that for a positive (resp. negative) argument the function returns a negative (resp. positive) value. With our system we will have for *neg* the type $(x : \{\nu \geq 0\} \rightarrow \{\nu \leq 0\}) \cap (x : \{\nu \leq 0\} \rightarrow \{\nu \geq 0\})$.

We introduce *Liquid Intersection Types*, a refinement type system with the addition of intersection types [2, 3]. Our use of intersection in conjunction with refinement types is motivated by a problem clearly identified for Liquid Types: the absence of most-general types, as in the ML tradition. Our use of intersection for refinement types draws some inspiration from [6], since this offers a mean to use jointly detailed types and intersections. Though, integrating this expressiveness with refinement types and keeping the qualifiers from $\mathbb{Q}$ simple (which must be provided by the programmer) implies the design of a new type system.

Besides the new type system, another contribution of this work is a new inference algorithm for Liquid Intersection Types.

This paper is organized as follows. Section 2 presents the designed type system, with a focus on the language syntax, semantics and typing rules, as well as a soundness result. The type inference algorithm is introduced in section 3. Finally, in section 4 we conclude with final remarks and explain some possible future work.

2 Type system

2.1 Syntax and semantics

M, N	$::=$		*Terms:*
	$\mid$	x	*variable*
	$\mid$	c	*constant*
	$\mid$	$\lambda x.M$	*abstraction*
	$\mid$	MN	*application*
	$\mid$	let $x = M$ in N	*let-binding*
	$\mid$	$[\Lambda\alpha]M$	*type abstraction*
	$\mid$	$[\tau]M$	*type instantiation*
ϕ	$::=$		*Refinements:*
	$\mid$	E	*arbitrary expressions*
	$\mid$	q	*qualifier from* $\mathbb{Q}$
	$\mid$	$\top$	*true (empty refinement)*
B	$::=$		*Base types :*
	$\mid$	`int`	*integers*
	$\mid$	`bool`	*booleans*
$\widetilde{\sigma}$	$::=$		*Liquid intersection pretypes :*
	$\mid$	$\{v : B \mid \phi\}$	*base refined type*
	$\mid$	$x : \widetilde{\sigma}_1 \to \widetilde{\sigma}_2$	*function*
	$\mid$	$\widetilde{\sigma}_1 \cap \widetilde{\sigma}_2$	*intersection*
	$\mid$	α	*type variable*
	$\mid$	$\forall\alpha.\widetilde{\sigma}$	*type schema*
τ	$::=$		*Simple types:*
	$\mid$	B	*basic type*
	$\mid$	α	*type variable*
	$\mid$	$\tau_1 \to \tau_2$	*functional type*
σ	$::=$		*Liquid Intersection Types :*
	$\mid$	$\widetilde{\sigma} :: \tau$	*well-founded pretype*
Γ	$::=$		*Environment:*
	$\mid$	$\emptyset$	*empty*
	$\mid$	$\Gamma; x : \sigma$	*new binding*

Figure 1: Syntax

Our target language is the λ-calculus extended with constants and local bindings via the `let` constructor. We assume the Barendregt convention regarding names of free and bound variables [1], and identify terms modulo α-equivalence. The syntax of expressions and types is presented in Figure 1.

The set of constants of our language is a countable alphabet of constants c, including literals and primitive functions. We assume for primitive functions the existence of at least arithmetic operators, a fixpoint combinator `fix` and an identifier representing `if-then-else` expressions. The type of

$$
\begin{array}{rcll}
v & ::= & & \textit{Values:} \\
 & | & x & \textit{variable} \\
 & | & \mathrm{c} & \textit{constant} \\
 & | & \lambda x.M & \textit{abstraction} \\
\textbf{Contexts} & & & \boxed{C} \\
C & ::= & & \textit{Contexts}: \\
 & | & [\,] & \textit{hole} \\
 & | & C\ x & \textit{left application} \\
 & | & \text{let } x = C \text{ in } M & \textit{let-context} \\
\textbf{Evaluation} & & & \boxed{M \rightsquigarrow N} \\
\mathrm{c}\ v & \rightsquigarrow & [\![\mathrm{c}]\!]\,(v) & [\mathcal{E} - \textit{Constant}] \\
(\lambda x.M)y & \rightsquigarrow & [y/x]M & [\mathcal{E} - \beta] \\
\text{let } x = v \text{ in } M & \rightsquigarrow & [v/x]M & [\mathcal{E} - \textit{Let}] \\
C[M] & \rightsquigarrow & C[N]\ \textit{if}\ M \rightsquigarrow N & [\mathcal{E} - \textit{Compat}]
\end{array}
$$

Figure 2: Small-step operational semantics

constants is established using a mapping $ty(c)$, assigning a refined type that captures the semantic of each constant. For instance, to an integer literal n it would be assigned the type $\{\nu : int \mid \nu = n\}$.

We use $\widetilde{\sigma}$ to denote pretypes (this notion of pretypes goes back to [11]), which stand for type variables, basic and functional refined types and intersection of pretypes. The notation $x : \tau_1 \to \tau_2$ will be preferred over the usual $\Pi(x : \tau_1).\tau_2$ for functional dependent types, meaning that variable x may occur in the refinement expressions present in τ_2. An intersection in pretypes (denoted by $\cap$) indicates that a term with type $\widetilde{\sigma}_1 \cap \widetilde{\sigma}_2$ has both type $\widetilde{\sigma}_1$ and $\widetilde{\sigma}_2$, respecting the possible refinement predicates figuring in these types.

A Liquid Intersection Type is a pretype $\widetilde{\sigma}$ for such that $\widetilde{\sigma} :: \tau$, for some τ (τ stands for simple types in the rest of the paper). The *well-founded* relation $\widetilde{\sigma} :: \tau$ is inductively defined by:

$$
\frac{}{\alpha :: \alpha}\ ::\text{-Var} \qquad
\frac{\widetilde{\sigma}_1 :: \tau \qquad \widetilde{\sigma}_2 :: \tau'}{\widetilde{\sigma}_1 \to \widetilde{\sigma}_2 :: \tau \to \tau'}\ ::\text{-Fun} \qquad
\frac{}{\{\nu : B \mid \phi\} :: B}\ ::\text{-Ref} \qquad
\frac{\widetilde{\sigma} :: \tau}{\forall \alpha . \widetilde{\sigma} :: \tau}\ ::\forall \qquad
\frac{\widetilde{\sigma}_i :: \tau\ (\forall i.\ 1 \le i \le n)}{\widetilde{\sigma}_1 \cap \ldots \cap \widetilde{\sigma}_n :: \tau}\ ::\cap
$$

Using this relation guarantees that intersection of types are at the refinement expressions only, i.e. for $\sigma_1 \cap \sigma_2$ both σ_1 and σ_2 are of the same form, solely differing in the refinement predicates.

To describe the execution behavior of our language we use a small-step contextual operational semantics, whose rules are shown in Figure 2.

A program written in our language can present arbitrary terms at application, but at run-time we restrict the arguments of applications to be variables and so sub-expressions are translated to nested let-bindings. This transformation is closely related with A-Normal Forms [5] and is performed to force types of intermediate expressions to be pushed into the typing context, so they can be used at the moment of application.

The relation $M \rightsquigarrow N$ describes a single evaluation step from term M to N. The rules $[\mathcal{E} - \beta]$, $[\mathcal{E} - \textit{Let}]$ and $[\mathcal{E} - \textit{Compat}]$ are standard for a call-by-value ML-like language (with the subtlety of variables in application). The rule $[\mathcal{E}\text{-}\textit{Constant}]$ evaluates an application with a constant in the function position. This rule relies on the embedding $[\![\cdot]\!]$ of terms into a decidable logic [10] (the definition of this embedding, as well as the details of the used logic, will be made clear in next section).

2.2 Typing rules

We present our typing rules via the collection of derivation rules shown in Figure 3. We present three different judgments: **type judgment**, of the form $\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma$ meaning that term M has Liquid Intersection Type σ under environment Γ, restricted to the qualifiers contained in $\mathbb{Q}$ i.e., only

expressions from the set $\mathbb{Q}$ can be used as refinement predicates for the following expressions: let bindings, λ-abstractions and type instantiations (for the sack of simplicity, we use σ to represent both types with refinements from $\mathbb{Q}$ and those with arbitrary refinements); **subtype judgment** $\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2$, stating that σ_1 is a subtype of σ_2 under the conditions of environment Γ; and the **well-formedness judgment** $\Gamma \vdash^{\cap} \sigma$ indicating that variables referred by the refinements of σ are in the scope of corresponding expressions. The well-formedness judgment can be lifted to well-formedness of environments, by stating that an environment is well-formed if for every binding types are well-formed with respect to the prefix environment. This well-formedness restriction implies the absence of the structural property of exchange in our system, since by permuting the bindings in Γ one could generate an inconsistent environment.

The rule [App] conforms to the dependent types discipline, since the type of an application MN is the return type of M but with every occurrence of x in the refinements substituted by N.

Another point worth mentioning is the distinction made when the type of a variable is to be retrieved, rules [Var-B] and [Var]. Whenever the type of the variable z is an intersection of refined basic type we ignore this refinements and assign z the type $\{\nu : B \mid \nu = z\}$, for some basic type B. This is inspired on the system of Liquid types [12], since this assigned refined type is very useful when it comes to use in subtyping, especially with the rule [$\prec$-Base].

One novel aspect of this system is the presence of the [Intersect] rule, which allows to intersect two types that have been derived for the same term. The use of this rule increases the expressiveness of the types language itself, since more detailed types can be derived for a program.

The subtyping relation presents some typical rules for a system with intersection types. These allow to capture the relations at the level of intersections in types, with no concern for the refinements of the two types being compared. On the other side, comparing two refined base types reduces to the check of an implication formula between the refinement expressions. Our system uses a decidable notion of implication in the rule [$\prec$-Base], by embedding environments and refinement expressions into a decidable logic. This logic contains at least equality, uninterpreted functions and linear arithmetic. The embedding $[\![M]\!]$ translates the term M to the correspondent one in the logic (if it is the case M is a constant or an arithmetic operator), or if M is a λ-abstraction or an application encodes it via uninterpreted functions. The embedding of environments is defined as

$$[\![\Gamma]\!] \triangleq \bigwedge \{([\![\phi_1]\!] \wedge \ldots \wedge [\![\phi_n]\!])[x/\nu] \mid x : \{\nu : B \mid \phi_1\} \cap \ldots \cap \{\nu : B \mid \phi_n\} \in \Gamma\}$$

Given that every implication expression generated in rule [$\prec$-Base] is decidable, it is then suitable to be discharged by some automatic theorem prover, like an SMT. So, type-checking in our system can be seen as a *typing-and-proof* process.

We show and example of a derivation for the term $\lambda x. -x$. With $\Gamma = x : \{\nu \geq 0\}$, consider:

$$\mathcal{D}_1' : \quad \dfrac{\textsc{Const}\ \dfrac{}{\Gamma \vdash^{\cap}_{\mathbb{Q}} - : (y : int \to \{\nu = -y\})} \qquad \textsc{Sub}\ \dfrac{\textsc{Var-B}\ \dfrac{\Gamma(x) = \{\nu \geq 0\}}{\Gamma \vdash^{\cap}_{\mathbb{Q}} x : \{\nu = x\}} \qquad \dfrac{\text{Valid}(x \geq 0 \wedge \nu = x \Rightarrow \top)}{\Gamma \vdash^{\cap} \{\nu = x\} \prec int}\ \prec\textsc{-Base}}{\Gamma \vdash^{\cap}_{\mathbb{Q}} x : int}}{\Gamma \vdash^{\cap}_{\mathbb{Q}} -x : \{\nu = -x\}}$$

and:

$$\mathcal{D}_1 : \quad \dfrac{\dfrac{\mathcal{D}_1' \qquad \dfrac{\text{Valid}(x \geq 0 \wedge \nu = -x \Rightarrow \nu \leq 0)}{\Gamma \vdash^{\cap} \{\nu = -x\} \prec \{\nu \leq 0\}}\ \prec\textsc{-Base}}{\Gamma \vdash^{\cap}_{\mathbb{Q}} -x : \{\nu \leq 0\}}\ \textsc{Sub}}{\vdash^{\cap}_{\mathbb{Q}} \lambda x. -x : (x : \{\nu \geq 0\} \to \{\nu \leq 0\})}\ \textsc{Fun}$$

We can also derive $\vdash^{\cap}_{\mathbb{Q}} \lambda x. -x : (x : \{\nu \leq 0\} \to \{\nu \geq 0\})$ (similarly to the previous derivation, with the corresponding $\leq$ and $\geq$ symbols changed). Naming that derivation $\mathcal{D}_2$, we finally have:

Liquid Intersection Type system $\boxed{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma}$

$$\textsc{Sub}\ \frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma' \qquad \Gamma \vdash^{\cap} \sigma' \prec \sigma \qquad \Gamma \vdash^{\cap} \sigma}{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma} \qquad \textsc{Intersect}\ \frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma \qquad \Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma' \qquad \sigma \cap \sigma' :: \tau}{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma \cap \sigma'}$$

$$\textsc{Var-B}\ \frac{\Gamma(x) = \sigma_1 \cap \ldots \cap \sigma_n \qquad \sigma_i :: B\,(\forall i : 1 \leq i \leq n)}{\Gamma \vdash^{\cap}_{\mathbb{Q}} x : \{v : B | v = x\}} \qquad \textsc{Var}\ \frac{\Gamma(x) \text{ not a base type} \qquad \Gamma(x) :: \tau}{\Gamma \vdash^{\cap}_{\mathbb{Q}} x : \Gamma(x)}$$

$$\textsc{App}\ \frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : (x : \sigma' \to \sigma) \qquad \Gamma \vdash^{\cap}_{\mathbb{Q}} N : \sigma'}{\Gamma \vdash^{\cap}_{\mathbb{Q}} MN : [N/x]\sigma} \qquad \textsc{Fun}\ \frac{\Gamma; x : \sigma \vdash^{\cap}_{\mathbb{Q}} M : \sigma' \qquad \Gamma \vdash^{\cap} \sigma \qquad \sigma :: \tau}{\Gamma \vdash^{\cap}_{\mathbb{Q}} \lambda x.M : \sigma \to \sigma'}$$

$$\textsc{Const}\ \frac{}{\Gamma \vdash^{\cap}_{\mathbb{Q}} \mathrm{c} : ty(\mathrm{c})} \qquad \textsc{Let}\ \frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma' \qquad \Gamma; x : \sigma' \vdash^{\cap}_{\mathbb{Q}} N : \sigma \qquad \Gamma \vdash^{\cap} \sigma}{\Gamma \vdash^{\cap}_{\mathbb{Q}} \text{let } x = M \text{ in } N : \sigma}$$

$$\textsc{Gen}\ \frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma \qquad \alpha \notin \Gamma}{\Gamma \vdash^{\cap}_{\mathbb{Q}} [\Lambda\alpha]M : \forall\alpha.\sigma} \qquad \textsc{Inst}\ \frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \forall\alpha.\sigma \qquad \Gamma \vdash^{\cap} \sigma' \qquad \text{Shape}(\sigma') = \tau}{\Gamma \vdash^{\cap}_{\mathbb{Q}} [\tau]M : [\sigma'/\alpha]\sigma}$$

Subtyping $\boxed{\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2}$

$$\prec\textsc{-Base}\ \frac{\texttt{Valid}([\![\Gamma]\!] \wedge ([\![\phi_1]\!] \wedge \ldots \wedge [\![\phi_n]\!]) \Rightarrow ([\![\phi_{n+1}]\!] \wedge \ldots \wedge [\![\phi_{n+m}]\!]))}{\Gamma \vdash^{\cap} \{v : B \mid \phi_1\} \cap \ldots \cap \{v : B \mid \phi_n\} \prec \{v : B \mid \phi_{n+1}\} \cap \ldots \cap \{v : B \mid \phi_{n+m}\}}$$

$$\prec\textsc{-Intersect-Fun}\ \frac{}{\Gamma \vdash^{\cap} (x : \sigma \to \sigma_1) \cap (x : \sigma \to \sigma_2) \prec (x : \sigma \to \sigma_1 \cap \sigma_2)}$$

$$\prec\textsc{-Fun}\ \frac{\Gamma \vdash^{\cap} \sigma_1' \prec \sigma_1 \qquad \Gamma; x : \sigma_1' \vdash^{\cap} \sigma_2 \prec \sigma_2'}{\Gamma \vdash^{\cap} x : \sigma_1 \to \sigma_2 \prec x : \sigma_1' \to \sigma_2'} \qquad \prec\textsc{-Var}\ \frac{}{\Gamma \vdash^{\cap} \alpha \prec \alpha}$$

$$\prec\textsc{-Left}\ \frac{}{\Gamma \vdash^{\cap} \sigma_1 \cap \sigma_2 \prec \sigma_1} \qquad \prec\textsc{-Right}\ \frac{}{\Gamma \vdash^{\cap} \sigma_1 \cap \sigma_2 \prec \sigma_2}$$

$$\prec\textsc{-Intersect}\ \frac{\Gamma \vdash^{\cap} \sigma \prec \sigma_1 \qquad \Gamma \vdash^{\cap} \sigma \prec \sigma_2}{\Gamma \vdash^{\cap} \sigma \prec \sigma_1 \cap \sigma_2} \qquad \textsc{Poly}\ \frac{\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2}{\Gamma \vdash^{\cap} \forall\alpha.\sigma_1 \prec \forall\alpha.\sigma_2}$$

Well formed types $\boxed{\Gamma \vdash^{\cap} \sigma}$

$$\text{WF-B}\ \frac{\Gamma; \nu : B \vdash^{\cap} \phi : \mathit{bool}}{\Gamma \vdash^{\cap} \{\nu : B \mid \phi\}} \qquad \text{WF-Var}\ \frac{}{\Gamma \vdash^{\cap} \alpha}$$

$$\text{WF-Fun}\ \frac{\Gamma; x : \sigma_1 \vdash^{\cap} \sigma_2}{\Gamma \vdash^{\cap} \sigma_1 \to \sigma_2} \qquad \text{WF-Poly}\ \frac{\Gamma \vdash^{\cap} \sigma}{\Gamma \vdash^{\cap} \forall\alpha.\sigma}$$

$$\text{WF-Intersect}\ \frac{\Gamma \vdash^{\cap} \sigma_1 \qquad \Gamma \vdash^{\cap} \sigma_2}{\Gamma \vdash^{\cap} \sigma_1 \cap \sigma_2}$$

Figure 3: Typing rules

$$\frac{\mathcal{D}_1 \qquad \mathcal{D}_2}{\vdash^{\cap}_{\mathbb{Q}} \lambda x.-x : (x : \{\nu \geq 0\} \to \{\nu \leq 0\}) \cap (x : \{\nu \leq 0\} \to \{\nu \geq 0\})}\ \textsc{Intersect}$$

We omit the well-formedness and well-founded sub-derivations, since they are trivially constructed.

2.3 Subject reduction

In order to prove soundness properties for our system we follow the approach of [12, 14]. The decidable notion of implication checking employed by the subtyping rules is a problem when it comes to prove a substitution lemma. So, instead we prove subject reduction for a version of our system with undecidable subtyping and unrestricted expressions in refinement predicates. The typing judgment in this system will be denoted by $\Gamma \vdash^{\cap} M : \sigma$. Then, we show that any derivation in the decidable system has a counter-part in the undecidable one. These results can be formalized as:

Theorem 1 (Subject Reduction)**.**

- *(Overapproximation) If* $\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma$ *then* $\Gamma \vdash^{\cap} M : \sigma$*;*
- *(Subject reduction) If* $\Gamma \vdash^{\cap} M : \sigma$ *and* $M \rightsquigarrow N$ *then* $\Gamma \vdash^{\cap} N : \sigma$*.*

The detailed proofs can be found in the appendix.

Combining the two previous results guarantees that at run-time, for every well-typed term, taking an evaluation step preserves types.

3 Type inference

We present in this section an algorithm for inferring Liquid Intersection Types, shown in Figure 4. The algorithm we propose is built upon three main phases: **(i)** we use the ML inference engine to get appropriate types, serving as *type shapes* for Liquid Intersection Types; **(ii)** for some particular sub-terms a set of constraints is generated, ensuring the well-formedness of types and that subtyping relations hold, in order to infer sound types; **(iii)** taking qualifiers from $\mathbb{Q}$ we solve the generated constraints *on-the-fly*, much like in classical inference algorithms.

$$
\begin{array}{lcl}
\texttt{Infer}(\Gamma, x, \mathbb{Q}) & = & \text{if } \mathcal{W}(\texttt{Shape}(\Gamma), x) = B \text{ then } \{v : B \mid v = x\} \\
 & & \text{else } \Gamma(x) \\
\texttt{Infer}(\Gamma, c, \mathbb{Q}) & = & ty(c) \\
\texttt{Infer}(\Gamma, \lambda x.M, \mathbb{Q}) & = & \text{let } (x : \sigma_1 \to \sigma'_1) \cap \ldots \cap (x : \sigma_n \to \sigma'_n) = \texttt{Fresh}(\mathcal{W}(\texttt{Shape}(\Gamma), \lambda x.M), \mathbb{Q}) \text{ in} \\
 & & \text{let } \sigma''_i = \texttt{Infer}(\Gamma; x : \sigma_i, M, \mathbb{Q}) \text{ in} \\
 & & \text{let } \mathcal{A} = \bigcap \left\{(x : \sigma_j \to \sigma'_j) \mid \Gamma \vdash^{\cap} (x : \sigma_1 \to \sigma'_1) \cap \ldots \cap (x : \sigma_n \to \sigma'_n)\right\} \text{ in} \\
 & & \bigcap \left\{(x : \sigma_k \to \sigma'_k) \mid x : \sigma_k \to \sigma'_k \in \mathcal{A}, \Gamma; x : \sigma_k \vdash^{\cap}_{\mathbb{Q}} \sigma''_k \prec \sigma'_k\right\} \\
\texttt{Infer}(\Gamma, MN, \mathbb{Q}) & = & \text{let } (x : \sigma_1 \to \sigma'_1) \cap \ldots \cap (x : \sigma_n \to \sigma'_n) = \texttt{Infer}(\Gamma, M, \mathbb{Q}) \text{ in} \\
 & & \text{let } \sigma = \texttt{Infer}(\Gamma, N, \mathbb{Q}) \text{ in} \\
 & & \bigcap [N/x] \left\{\sigma'_i \mid \Gamma \vdash^{\cap}_{\mathbb{Q}} \sigma \prec \sigma_i\right\} \\
\texttt{Infer}(\Gamma, \text{let } x = M \text{ in } N, \mathbb{Q}) & = & \text{let } \sigma = \texttt{Fresh}(\mathcal{W}(\texttt{Shape}(\Gamma), \text{let } x = M \text{ in } N), \mathbb{Q}) \text{ in} \\
 & & \text{let } \sigma_1 = \texttt{Infer}(\Gamma, M, \mathbb{Q}) \text{ in} \\
 & & \text{let } \sigma_2 = \texttt{Infer}(\Gamma; x : \sigma_1, N, \mathbb{Q}) \text{ in} \\
 & & \text{let } \mathcal{A} = \bigcap \{\sigma'_i \mid \Gamma \vdash^{\cap} \sigma\} \text{ in} \\
 & & \bigcap \left\{\sigma'' \mid \sigma'' \in \mathcal{A}, \Gamma; x : \sigma_1 \vdash^{\cap}_{\mathbb{Q}} \sigma_2 \prec \sigma''\right\} \\
\texttt{Infer}(\Gamma, [\Lambda\alpha]M, \mathbb{Q}) & = & \text{let } \sigma = \texttt{Infer}(\Gamma, M, \mathbb{Q}) \text{ in} \\
 & & \forall\alpha.\sigma \\
\texttt{Infer}(\Gamma, [\tau]M, \mathbb{Q}) & = & \text{let } \tau' = \texttt{Fresh}(\tau, \mathbb{Q}) \text{ in} \\
 & & \text{let } \forall\alpha.\sigma = \texttt{Infer}(\Gamma, M, \mathbb{Q}) \text{ in} \\
 & & \texttt{let } \mathcal{A} = \bigcap \{\tau'_i \mid \Gamma \vdash^{\cap} \tau'\} \text{ in} \\
 & & \sigma[\mathcal{A}/\alpha]
\end{array}
$$

Figure 4: Type inference algorithm

3.1 Using Damas-Milner type inference

One key aspect of our inference algorithm is the use of the inference algorithm $\mathcal{W}$ [4] to infer ML types. Given the fact that a Liquid Intersection Type for a term is a refinement and intersections of the corresponding ML type, the types inferred by $\mathcal{W}$ act as *shapes* for our Liquid Intersection Types. Indeed, the function $\texttt{Shape}(\cdot)$ (figuring in the typing rules and in the inference algorithm) maps a Liquid Intersection Type to its corresponding ML type. For example, $\texttt{Shape}((x : \{\nu = 0\} \to \{\nu = 0\}) \cap (x : \{\nu \geq 0\} \to \{\nu \geq 0\})) = int \to int$.

In the inference algorithm, whenever $\mathcal{W}$ is called, we need to feed it with an environment containing exclusively ML types. This is done by lifting $\texttt{Shape}(\cdot)$ to environments, $\texttt{Shape}(\Gamma)$, by applying it to every bindings in Γ.

The function $\texttt{Fresh}(\cdot, \cdot)$ takes an ML type and the set $\mathbb{Q}$ as input and generates a new Liquid Intersection Type that contains all the combinations of refinement expressions from $\mathbb{Q}$. Taking for instance the ML type $\tau = x : int \to int$ (we assume we can annotate types with the corresponding abstraction variable, so it is easier to use with refinements) and $\mathbb{Q} = \{\nu \geq 0, \nu \leq 0\}$, $\texttt{Fresh}(\tau, \mathbb{Q})$ would generate the Liquid Intersection Type

$$
\begin{array}{l}
(x : \{\nu \geq 0\} \to \{\nu \geq 0\}) \cap \\
(x : \{\nu \geq 0\} \to \{\nu \leq 0\}) \cap \\
(x : \{\nu \leq 0\} \to \{\nu \geq 0\}) \cap \\
(x : \{\nu \leq 0\} \to \{\nu \leq 0\})
\end{array}
$$

3.2 Constraint generation

The constraints generated during inference serve as a means to ensure that the subtyping and well-formedness requirements are respected. In the presentation of the algorithm we borrow the notations from the typing rules, with $\Gamma \vdash^{\cap} \sigma$ standing for a well-formedness restriction over σ and $\Gamma \vdash^{\cap} \sigma \prec \sigma'$ constraining type σ to be a subtype of σ'.

The well-formedness constraints are generated for terms where a fresh Liquid Intersection Type is generated (λ-abstractions, let-bindings and type application). For a fresh generated Liquid Intersection Type, solving this kind of constraints will result in a type where the free-variables of every refinement are in scope of the corresponding expression.

The second class of constraints are the subtyping constraints, capturing relations between two Liquid Intersection Types. A constraint $\Gamma \vdash^{\cap} \sigma \prec \sigma'$ is valid if the type σ' is a super-type of σ, meaning that there is a type derivation using the subsumption rule to relate the two types.

The well-formedness and subtyping rules (Figure 3) can be used to simplify constraints prior to their solving. For instance, the constraint $\Gamma \vdash^{\cap} \sigma_1 \cap \ldots \cap \sigma_n$ can be simplified to the set $\{\Gamma \vdash^{\cap} \sigma_1, \ldots, \Gamma \vdash^{\cap} \sigma_n\}$. On the other hand, the constraint $\Gamma \vdash^{\cap} x : \sigma_1 \to \sigma_2 \prec x : \sigma_1' \to \sigma_2'$ can be further reduced to $\Gamma \vdash^{\cap} \sigma_1' \prec \sigma_1$ and $\Gamma; x : \sigma_1' \vdash^{\cap} \sigma_2 \prec \sigma_2'$.

3.3 Constraint solving

We now describe the process of solving the collected constraints throughout the inference algorithm. This process will reduce to two different validity tests: a well-formedness constraint will, ultimately, reduce to the constraint of the form $\Gamma \vdash^{\cap} \{\nu : B \mid \phi\}$ and so it will amount to check if the type `bool` can be derived for ϕ under Γ; for the subtyping case, the simplification of constraints will result in a series of restrictions of the form $\Gamma \vdash^{\cap} \{\nu : B \mid \phi_1\} \cap \ldots \cap \{\nu : B \mid \phi_n\} \prec \{\nu : B \mid \phi_1'\} \cap \ldots \cap \{\nu : B \mid \phi_m'\}$, leading to check if $[\![\Gamma]\!] \wedge [\![\phi_1]\!] \wedge \ldots \wedge [\![\phi_n]\!] \Rightarrow [\![\phi_1']\!] \wedge \ldots \wedge [\![\phi_m']\!]$ holds.

Whenever well-formedness constraints are generated, these are solved before the subtyping ones. This step ensures only well-formed types are involved on subtyping relations. Well-formedness constraints arise when a *fresh* Liquid Intersection Type is generated, since that is when refinement expressions are plugged into a type. Such fresh types will be of the form $\sigma_1 \cap \ldots \cap \sigma_n$, so the solution for a constraint of the form $\Gamma \vdash^{\cap} \sigma_1 \cap \ldots \cap \sigma_n$ is the type $\bigcap \{\sigma_i\}$, the intersection of all σ_i (with $1 \leq i \leq n$) such that $\Gamma \vdash^{\cap} \sigma_i$. We assign this solution to a *temporary* type, denoted by $\mathcal{A}$, which will be used during the solving of subtyping constraints.

The subtyping constraints will ensure that inferred types only present refinement expressions capturing the functional behavior of terms. These will be used with λ-abstractions, applications and let-bindings. Except for applications, subtyping constraints are preceded by the resolution of well-formedness restrictions, and so it is the case that subtyping relations will be checked using the temporary type $\mathcal{A}$.

For the case of λ-abstractions, after generating the fresh Liquid Intersection Type $(x : \sigma_1 \to \sigma'_1) \cap \ldots \cap (x : \sigma_n \to \sigma'_n)$, a series of calls to `Infer` are triggered, which we present via the syntax let $\sigma''_i = \texttt{Infer}(\Gamma; x : \sigma_i, M, \mathbb{Q})$, with $1 \leq i \leq n$. This calls differ only on the type σ_i of x pushed into the environment, implying that different types for M can be inferred. After solving the well-formedness constraints, we must remove from type $\mathcal{A}$ the refinement expressions that would cause the type to be unsound. We use the notation $x : \sigma_k \to \sigma'_k \in \mathcal{A}$ to indicate that $\bigcap\{x : \sigma_k \to \sigma'_k\}$ should be a supertype of $\mathcal{A}$, in the sense that it can be obtained from $\mathcal{A}$ using exclusively the rules [$\prec$-Left] or [$\prec$-Right] (taking an analogy with set theory, $\bigcap\{x : \sigma_k \to \sigma'_k\}$ would be a sub-set of the intersections of $\mathcal{A}$). Then, the inferred type will be $\bigcap\{x : \sigma_k \to \sigma'_k\}$, such that $x : \sigma_k \to \sigma'_k \in \mathcal{A}$ and the constraint $\Gamma; x : \sigma_k \vdash^{\cap} \sigma''_k \prec \sigma'_k$ is valid, that is, the type inferred for M under the environment $\Gamma; x : \sigma_k$ is a subtype of σ'_k. As an example, consider $\mathbb{Q} = \{\nu \geq 0, \nu \leq 0, y = 5\}$, the term $\lambda x. -x$ and $\Gamma = \emptyset$. The inference procedure will start by generating the type:

$$
\begin{array}{l}
(x : \{\nu \geq 0\} \to \{\nu \geq 0\}) \cap \\
(x : \{\nu \geq 0\} \to \{\nu \leq 0\}) \cap \\
(x : \{\nu \leq 0\} \to \{\nu \geq 0\}) \cap \\
(x : \{\nu \leq 0\} \to \{\nu \leq 0\}) \cap \\
(x : \{\nu \geq 0\} \to \{y = 5\}) \cap \\
(x : \{\nu \leq 0\} \to \{y = 5\}) \cap \\
(x : \{y = 5\} \to \{\nu \geq 0\}) \cap \\
(x : \{y = 5\} \to \{\nu \leq 0\}) \cap \\
(x : \{y = 5\} \to \{y = 5\})
\end{array}
$$

Then, with well-formedness constraints, and since no variable y is in scope, we are left with:

$$
\begin{array}{l}
(x : \{\nu \geq 0\} \to \{\nu \geq 0\}) \cap \\
(x : \{\nu \geq 0\} \to \{\nu \leq 0\}) \cap \\
(x : \{\nu \leq 0\} \to \{\nu \geq 0\}) \cap \\
(x : \{\nu \leq 0\} \to \{\nu \leq 0\})
\end{array}
$$

Finally, because of subtyping relations, the inferred type will be:

$$
\begin{array}{l}
(x : \{\nu \geq 0\} \to \{\nu \leq 0\}) \cap \\
(x : \{\nu \leq 0\} \to \{\nu \geq 0\})
\end{array}
$$

For application and let-bindings, solving subtyping constraints works in a similar manner as for λ-abstractions. The type of an application is inferred similarly as in [6]: for the function M with type $x : \sigma_1 \to \sigma'_1 \cap \ldots \cap \sigma_n \to \sigma'_n$ and the argument N with type σ, the type of MN is $\bigcap\{\sigma'_i\}$, such that $1 \leq i \leq n$ and $\Gamma \vdash^{\cap} \sigma \prec \sigma_i$ is checked valid.

3.4 Properties of inference

We were able to prove that our inference algorithm is sound with respected to the type rules. This property is formalized as follows:

Theorem 2 (Soundness)**.** *If* $\text{Infer}(\Gamma, M, \mathbb{Q}) = \sigma$ *then* $\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma$.

4 Conclusion and future work

We present a new type system supporting functional descriptions, via refinement types, and offering the expressiveness of intersection types. This type system can be used to derive more precise types than in previous refinement type systems.

To design a decidable system we adopted a style closely related to Liquid Types: the refinement expressions presented in types are exclusively collected from $\mathbb{Q}$, a global set of logical qualifiers, and the subtyping is decidable. We also impose that the type of an expression must the an intersections of refinements to its ML type, intersecting only types of the same form.

We also proposed an inference algorithm for Liquid Intersection Types. This algorithm takes as input an environment Γ, a term M and the set of qualifiers $\mathbb{Q}$, producing a correspondent Liquid Intersection Type. Our inference algorithm uses the $\mathcal{W}$ algorithm to infer the shape of a Liquid Intersection Type, which is the ML type for that term. To determine which refinement expressions can be plugged into a type, the algorithm produces a series of well-formedness and subtyping constraints, solving them immediately after their generation. We have been able to prove that our algorithm is sound with the respected to the conceived typing rules.

Current and future work includes the study of completeness of type inference for our system and to extend decidable intersection type systems (of finite ranks [7, 8]) with type refinement predicates.

References

[1] H. P. Barendregt. *The Lambda Calculus, its Syntax and Semantics, Revised second edition.* North-Holland, 1984.

[2] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The journal of symbolic logic*, 48(4):931–940, 1983.

[3] M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 10 1980.

[4] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '82, pages 207–212, New York, NY, USA, 1982. ACM.

[5] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation*, PLDI '93, pages 237–247, New York, NY, USA, 1993. ACM.

[6] Tim Freeman and Frank Pfenning. Refinement types for ML. In *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, PLDI '91, pages 268–277, New York, NY, USA, 1991. ACM.

[7] Trevor Jim. Rank 2 type systems and recursive definitions. *Massachusetts Institute of Technology, Cambridge, MA*, 1995.

[8] A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 161–174, New York, NY, USA, 1999. ACM.

[9] Kenneth Knowles and Cormac Flanagan. Hybrid type checking. *ACM Trans. Program. Lang. Syst.*, 32(2):6:1–6:34, February 2010.

[10] Charles Gregory Nelson. *Techniques for Program Verification.* PhD thesis, Stanford, CA, USA, 1980. AAI8011683.

[11] C.-H. Luke Ong and Takeshi Tsukada. Two-level game semantics, intersection types, and recursion schemes. In *Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, pages 325–336, Berlin, Heidelberg, 2012. Springer-Verlag.

[12] Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. Liquid types. In *Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, pages 159–169, New York, NY, USA, 2008. ACM.

[13] Robert E. Shostak. Deciding combinations of theories. *J. ACM*, 31(1):1–12, January 1984.

[14] Niki Vazou, Patrick M. Rondon, and Ranjit Jhala. Abstract refinement types. In *Proceedings of the 22Nd European Conference on Programming Languages and Systems*, ESOP'13, pages 209–228, Berlin, Heidelberg, 2013. Springer-Verlag.

A Correctness of Type Checking

Undecidable type system . An undecidable version of our type system is presented in figures 5 and 6. The main difference to the system in figure 3 is the use of undecidable subtyping and the fact refinements are arbitrary boolean formulas.

Definition 1 (Constants)**.** *Each constant c has a type ty(c) such that:*

1. $\emptyset \vdash^{\cap} ty(c)$*;*
2. *if c is a primitive function then it cannot get stuck, if* $\Gamma \vdash^{\cap} \mathrm{c}\ v$ *then* $[\![\mathrm{c}]\!](v)$ *is defined and if* $\Gamma \vdash^{\cap} \mathrm{c}\ M : \sigma$ *and* $[\![\mathrm{c}]\!](M)$ *is defined then* $\Gamma \vdash^{\cap} [\![\mathrm{c}]\!]\ (M) : \sigma$*;*
3. *if* $ty(c)$ *is* $\{\nu : B \mid \phi\}$ *then* $\phi \equiv \nu = c$*.*

Definition 2 (Embedding)**.** *The embedding* $[\![]\!]$ *is defined as a map from terms and environments to formulas in the decidable logic such that for all* Γ, ϕ, ϕ' *if* $\Gamma \vdash^{\cap} \phi : bool$*,* $\Gamma \vdash^{\cap} \phi' : bool$*,* $\mathrm{Valid}([\![\Gamma]\!] \wedge [\![\phi]\!] \Rightarrow \phi')$*, then* $\Gamma \vdash^{\cap} \phi \Rightarrow \phi'$*.*

Definition 3 (Substitution)**.** *We define substitution on types,* $\phi\sigma$*, as follows:*

$$\begin{array}{rcl} \rho(\alpha) & = & \alpha \\ \rho(\{v : B \mid \phi\}) & = & \{v : B \mid \rho\phi\} \\ \rho(x : \sigma_1 \to \sigma_2) & = & x : \rho\sigma_1 \to \rho\sigma_2 \\ \rho(\forall\alpha.\sigma) & = & \forall\alpha.(\rho\sigma) \\ \rho(\sigma_1 \cap \sigma_2) & = & (\rho\sigma_1) \cap (\rho\sigma_2) \end{array}$$

Lemma 1 (Weakening)**.** *If*

$$\begin{array}{l} \Gamma = \Gamma_1; \Gamma_2 \\ \Gamma' = \Gamma_1; x : \sigma_x; \Gamma_2 \end{array}$$

then:

1. *if* $\Gamma' \models \rho_1; [v/x]; \rho_2$ *then* $\Gamma \models \rho_1; \rho_2$*;*
2. *if* $\Gamma \vdash^{\cap} \phi \Rightarrow \phi'$ *then* $\Gamma' \vdash^{\cap} \phi \Rightarrow \phi'$*;*
3. *if* $\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2$ *then* $\Gamma' \vdash^{\cap} \sigma_1 \prec \sigma_2$*;*
4. *if* $\Gamma \vdash^{\cap} \sigma$ *then* $\Gamma' \vdash^{\cap} \sigma$*;*
5. *if* $\Gamma \vdash^{\cap} M : \sigma$ *then* $\Gamma' \vdash^{\cap} M : \sigma$*.*

Proof. By simultaneous induction on the derivations.

Liquid Intersection Type system $\boxed{\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma}$

$$\textsc{Sub}\ \frac{\Gamma \vdash^{\cap} M : \sigma' \qquad \Gamma \vdash^{\cap} \sigma' \prec \sigma \qquad \Gamma \vdash^{\cap} \sigma}{\Gamma \vdash^{\cap} M : \sigma} \qquad \textsc{Intersect}\ \frac{\Gamma \vdash^{\cap} M : \sigma \qquad \Gamma \vdash^{\cap} M : \sigma' \qquad \sigma \cap \sigma' :: \tau}{\Gamma \vdash^{\cap} M : \sigma \cap \sigma'}$$

$$\textsc{Var-B}\ \frac{\Gamma(x) = \sigma_1 \cap \ldots \cap \sigma_n \qquad \sigma_i :: B\,(\forall i : 1 \leq i \leq n)}{\Gamma \vdash^{\cap} x : \{v : B | v = x\}} \qquad \textsc{Var}\ \frac{\Gamma(x) \text{ not a base type} \qquad \Gamma(x) :: \tau}{\Gamma \vdash^{\cap} x : \Gamma(x)}$$

$$\textsc{App}\ \frac{\Gamma \vdash^{\cap} M : (x : \sigma' \to \sigma) \qquad \Gamma \vdash^{\cap} N : \sigma'}{\Gamma \vdash^{\cap} MN : [N/x]\sigma} \qquad \textsc{Fun}\ \frac{\Gamma; x : \sigma \vdash^{\cap} M : \sigma' \qquad \Gamma \vdash^{\cap} \sigma \qquad \sigma :: \tau}{\Gamma \vdash^{\cap} \lambda x.M : \sigma \to \sigma'}$$

$$\textsc{Const}\ \frac{}{\Gamma \vdash^{\cap} \mathrm{c} : ty(\mathrm{c})} \qquad \textsc{Let}\ \frac{\Gamma \vdash^{\cap} M : \sigma' \qquad \Gamma; x : \sigma' \vdash^{\cap} N : \sigma \qquad \Gamma \vdash^{\cap} \sigma}{\Gamma \vdash^{\cap} \text{let } x = M \text{ in } N : \sigma}$$

$$\textsc{Gen}\ \frac{\Gamma \vdash^{\cap} M : \sigma \qquad \alpha \notin \Gamma}{\Gamma \vdash^{\cap} [\Lambda\alpha]M : \forall\alpha.\sigma} \qquad \textsc{Inst}\ \frac{\Gamma \vdash^{\cap} M : \forall\alpha.\sigma \qquad \Gamma \vdash^{\cap} \sigma' \qquad \mathrm{Shape}(\sigma') = \tau}{\Gamma \vdash^{\cap} [\tau]M : [\sigma'/\alpha]\sigma}$$

Implication $\boxed{\Gamma \vdash^{\cap} \phi \Rightarrow \phi'}$

$$\textsc{Imp}\ \frac{\Gamma \vdash^{\cap} \phi : \mathit{bool} \qquad \Gamma \vdash^{\cap} \phi' : \mathit{bool} \qquad \forall\rho.(\Gamma \models \rho \text{ and } \rho\phi \overset{*}{\rightsquigarrow} \top \text{ implies } \rho\phi' \overset{*}{\rightsquigarrow} \top)}{\Gamma^{\cap}\phi \Rightarrow \phi'}$$

Subtyping $\boxed{\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2}$

$$\prec\text{-Base}\ \frac{\Gamma; \nu : B \vdash^{\cap} \phi_1 \wedge \cdots \wedge \phi_n \Rightarrow \phi'_1 \wedge \cdots \wedge \phi'_m}{\Gamma \vdash^{\cap} \{v : B \mid \phi_1\} \cap \ldots \cap \{v : B \mid \phi_n\} \prec \{v : B \mid \phi_{n+1}\} \cap \ldots \cap \{v : B \mid \phi_{n+m}\}}$$

$$\prec\text{-Intersect-Fun}\ \frac{}{\Gamma \vdash^{\cap} (x : \sigma \to \sigma_1) \cap (x : \sigma \to \sigma_2) \prec (x : \sigma \to \sigma_1 \cap \sigma_2)}$$

$$\prec\text{-Fun}\ \frac{\Gamma \vdash^{\cap} \sigma'_1 \prec \sigma_1 \qquad \Gamma; x : \sigma'_1 \vdash^{\cap} \sigma_2 \prec \sigma'_2}{\Gamma \vdash^{\cap} x : \sigma_1 \to \sigma_2 \prec x : \sigma'_1 \to \sigma'_2} \qquad \prec\text{-Var}\ \frac{}{\Gamma \vdash^{\cap} \alpha \prec \alpha}$$

$$\prec\text{-Left}\ \frac{}{\Gamma \vdash^{\cap} \sigma_1 \cap \sigma_2 \prec \sigma_1} \qquad \prec\text{-Right}\ \frac{}{\Gamma \vdash^{\cap} \sigma_1 \cap \sigma_2 \prec \sigma_2}$$

$$\prec\text{-Intersect}\ \frac{\Gamma \vdash^{\cap} \sigma \prec \sigma_1 \qquad \Gamma \vdash^{\cap} \sigma \prec \sigma_2}{\Gamma \vdash^{\cap} \sigma \prec \sigma_1 \cap \sigma_2} \qquad \textsc{Poly}\ \frac{\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2}{\Gamma \vdash^{\cap} \forall\alpha.\sigma_1 \prec \forall\alpha.\sigma_2}$$

Figure 5: Dependent Intersection typing rules

Well formed types $\boxed{\Gamma \vdash^{\cap} \sigma}$

$$\frac{\Gamma; \nu : B \vdash^{\cap} \phi : \mathit{bool}}{\Gamma \vdash^{\cap} \{\nu : B \mid \phi\}}\ \text{WF-B} \qquad \frac{}{\Gamma \vdash^{\cap} \alpha}\ \text{WF-Var}$$

$$\frac{\Gamma; x : \sigma_1 \vdash^{\cap} \sigma_2}{\Gamma \vdash^{\cap} \sigma_1 \to \sigma_2}\ \text{WF-Fun} \qquad \frac{\Gamma \vdash^{\cap} \sigma}{\Gamma \vdash^{\cap} \forall \alpha.\sigma}\ \text{WF-Poly}$$

$$\frac{\Gamma \vdash^{\cap} \sigma_1 \qquad \Gamma \vdash^{\cap} \sigma_2}{\Gamma \vdash^{\cap} \sigma_1 \cap \sigma_2}\ \text{WF-Intersect}$$

Consistent substitutions $\boxed{\Gamma \models \rho}$

$$\frac{}{\emptyset \models \emptyset}\ \text{CS-Empty} \qquad \frac{\Gamma \models \rho \qquad \emptyset \vdash^{\cap} v : \rho\sigma}{\Gamma; x : \sigma \models \rho; [v/x]}\ \text{CS-Ext}$$

Figure 6: Rules for well formed Intersection Dependent Types and consistent substitutions

1. Assume
$$\Gamma' \models \rho_1; [v/x]; \rho_2$$
By the definition of Γ' and the rules for consistent substitutions, we know
$$\Gamma_1 \models \rho_1$$
$$\emptyset \vdash^{\cap} v : \rho_1 \sigma_x$$
$$(\rho_1; [v/x])\Gamma_2 \models \rho_2$$
Since $x \notin \text{FreeVars}(\Gamma_2)$ (variables are bounded at most once in environments) we have $(\rho_1; [v/x])\Gamma_2 = \rho_1\Gamma_2$.
So, by repeated use of rule [CS-Ext],
$$\Gamma_1; \Gamma_2 \models \rho_1; \rho_2$$
Since $\Gamma = \Gamma_1; \Gamma_2$ it completes the proof.

2. Assume
$$\Gamma \vdash^{\cap} \phi \Rightarrow \phi'$$
By inversion on the rule [Imp], we have
$$\Gamma \vdash^{\cap} \phi : \mathit{bool} \tag{a}$$
$$\Gamma \vdash^{\cap} \phi' : \mathit{bool} \tag{b}$$
$$\forall \rho.(\Gamma \models \rho \text{ and } \rho\phi \overset{*}{\rightsquigarrow} \top \text{ implies } \rho\phi' \overset{*}{\rightsquigarrow} \top) \tag{c}$$
By IH (5.) we get
$$\Gamma' \vdash^{\cap} \phi : \mathit{bool}$$
$$\Gamma' \vdash^{\cap} \phi' : \mathit{bool}$$
We consider any ρ' such that $\Gamma' \models \rho'$ and $\rho'\phi \overset{*}{\rightsquigarrow} \top$. The substitution ρ' must be of the form $\rho_1; [v/x]; \rho_2$ and so $\rho \equiv \rho_1; \rho_2$.

From (a) and (b) we know $x \notin \text{FreeVars}(\phi) \cup \text{FreeVars}(\phi')$ so

$$\rho\phi = \rho'\phi$$
$$\rho\phi = \rho'\phi'$$

From (c) we then have

$$\forall \rho'.(\Gamma' \models \rho' \text{ and } \rho'\phi \overset{*}{\rightsquigarrow} \top \text{ implies } \rho'\phi' \overset{*}{\rightsquigarrow} \top)$$

which completes the proof.

3. By induction of the derivation of $\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2$, splitting cases on which rule was used at the bottom.

   - case [$\prec$-Base]: Assume
   $$\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2$$
   where $\sigma_1, \sigma_2 = \{\nu : B \mid \phi_1\} \cap \cdots \cap \{\nu : B \mid \phi_n\}, \{\nu : B \mid \phi'_1\} \cap \cdots \cap \{\nu : B \mid \phi'_m\}$.
   By inversion
   $$\Gamma; \nu : B \vdash^{\cap} \phi_1 \wedge \cdots \wedge \phi_n \Rightarrow \phi'_1 \wedge \cdots \wedge \phi'_m$$
   By IH (2.)
   $$\Gamma'; \nu : B \vdash^{\cap} \phi_1 \wedge \cdots \wedge \phi_n \Rightarrow \phi'_1 \wedge \cdots \wedge \phi'_m$$
   So, the following derivation is valid
   $$\prec\text{-Base}\ \frac{\Gamma'; \nu : B \vdash^{\cap} \phi_1 \wedge \cdots \wedge \phi_n \Rightarrow \phi'_1 \wedge \cdots \wedge \phi'_m}{\Gamma' \vdash^{\cap} \{\nu : B \mid \phi_1\} \cap \cdots \cap \{\nu : B \mid \phi_n\} \prec \{\nu : B \mid \phi'_1\} \cap \cdots \cap \{\nu : B \mid \phi'_m\}}$$

   - case [$\prec$-Intersect-Fun]: Easy, by the following derivation
   $$\prec\text{-Intersect-Fun}\ \frac{}{\Gamma' \vdash^{\cap} (x : \sigma \to \sigma_1) \cap (x : \sigma \to \sigma_2) \prec (x : \sigma \to \sigma_1 \cap \sigma_2)}$$

   - case [$\prec$-Fun]: By inversion
   $$\Gamma \vdash^{\cap} \sigma'_1 \prec \sigma_1$$
   $$\Gamma; x : \sigma'_1 \vdash^{\cap} \sigma_2 \prec \sigma'_2$$
   By IH
   $$\Gamma' \vdash^{\cap} \sigma'_1 \prec \sigma_1$$
   $$\Gamma'; x : \sigma'_1 \vdash^{\cap} \sigma_2 \prec \sigma'_2$$
   So, the following derivation is valid
   $$\prec\text{-Fun}\ \frac{\Gamma' \vdash^{\cap} \sigma'_1 \prec \sigma_1 \qquad \Gamma'; x : \sigma'_1 \vdash^{\cap} \sigma_2 \prec \sigma'_2}{\Gamma' \vdash^{\cap} x : \sigma_1 \to \sigma_2 \prec x : \sigma'_1 \to \sigma'_2}$$

   - case [$\prec$-Var]: Easy, by the following derivation
   $$\prec\text{-Var}\ \frac{}{\Gamma' \vdash^{\cap} \alpha \prec \alpha}$$

   - case [$\prec$-Left]: Easy, by the following derivation
   $$\prec\text{-Left}\ \frac{}{\Gamma' \vdash^{\cap} \sigma_1 \cap \sigma_2 \prec \sigma_1}$$

   - case [$\prec$-Right]: Easy, by the following derivation
   $$\prec\text{-Right}\ \frac{}{\Gamma' \vdash^{\cap} \sigma_1 \cap \sigma_2 \prec \sigma_2}$$

- case [$\prec$-Intersect]: By inversion

$$\Gamma \vdash^{\cap} \sigma \prec \sigma_1$$
$$\Gamma \vdash^{\cap} \sigma \prec \sigma_2$$

By IH

$$\Gamma' \vdash^{\cap} \sigma \prec \sigma_1$$
$$\Gamma' \vdash^{\cap} \sigma \prec \sigma_2$$

So, the following derivation in valid

$$\prec\text{-Intersect}\ \frac{\Gamma' \vdash^{\cap} \sigma \prec \sigma_1 \qquad \Gamma' \vdash^{\cap} \sigma \prec \sigma_2}{\Gamma' \vdash^{\cap} \sigma \prec \sigma_1 \cap \sigma_2}$$

- case [$\prec$-Poly]: By inversion

$$\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2$$

By IH

$$\Gamma' \vdash^{\cap} \sigma_1 \prec \sigma_2$$

So, the following derivation in valid

$$\prec\text{-Poly}\ \frac{\Gamma' \vdash^{\cap} \sigma_1 \prec \sigma_2}{\Gamma' \vdash^{\cap} \forall\alpha.\sigma_1 \prec \forall\alpha.\sigma_2}$$

4. By induction on the derivation of $\Gamma \vdash^{\cap} \sigma$, splitting cases on which rule was used at the bottom.

- case [WF-B]: By inversion

$$\Gamma; \nu : B \vdash^{\cap} \phi : \mathit{bool}$$

By IH (5.):

$$\Gamma'; \nu : B \vdash^{\cap} \phi : \mathit{bool}$$

So, the following derivation is valid

$$\text{WF-Intersect}\ \frac{\Gamma'; \nu : B \vdash^{\cap} \phi : \mathit{bool}}{\Gamma' \vdash^{\cap} \{\nu : B \mid \phi\}}$$

- case [WF-Var]: Easy, by the following derivation

$$\text{WF-Var}\ \frac{}{\Gamma' \vdash^{\cap} \alpha}$$

- case [WF-Fun]: By inversion

$$\Gamma; x : \sigma_1 \vdash^{\cap} \sigma_2$$

By IH

$$\Gamma'; x : \sigma_1 \vdash^{\cap} \sigma_2$$

So, the following derivation is valid

$$\text{WF-Fun}\ \frac{\Gamma'; x : \sigma_1 \vdash^{\cap} \sigma_2}{\Gamma' \vdash^{\cap} x : \sigma_1 \to \sigma_2}$$

- case [WF-Poly]: By inversion

$$\Gamma \vdash^{\cap} \sigma$$

By IH

$$\Gamma' \vdash^{\cap} \sigma$$

So, the following derivation is valid

$$\text{WF-Poly}\ \frac{\Gamma' \vdash^{\cap} \sigma}{\Gamma' \vdash^{\cap} \forall\alpha.\sigma}$$

- case [WF-Intersect]: By inversion

$$\Gamma \vdash^{\cap} \sigma_1$$
$$\Gamma \vdash^{\cap} \sigma_2$$

By IH

$$\Gamma' \vdash^{\cap} \sigma_1$$
$$\Gamma' \vdash^{\cap} \sigma_2$$

So, the following derivation is valid

$$\textsc{WF-Intersect}\ \frac{\Gamma' \vdash^{\cap} \sigma_1 \qquad \Gamma' \vdash^{\cap} \sigma_2}{\Gamma' \vdash^{\cap} \sigma_1 \cap \sigma_2}$$

5. By induction on the derivation of $\Gamma \vdash^{\cap} M : \sigma$, splitting cases on which rule was used at the bottom.

- case [Sub]: By inversion

$$\Gamma \vdash^{\cap} M : \sigma'$$
$$\Gamma \vdash^{\cap} \sigma' \prec \sigma$$
$$\Gamma \vdash \sigma$$

By IH, (3.) and (4.)

$$\Gamma' \vdash^{\cap} M : \sigma'$$
$$\Gamma' \vdash^{\cap} \sigma' \prec \sigma$$
$$\Gamma' \vdash^{\cap} \sigma$$

So, the following derivation is valid

$$\textsc{Sub}\ \frac{\Gamma' \vdash^{\cap} M : \sigma' \qquad \Gamma' \vdash^{\cap} \sigma' \prec \sigma \qquad \Gamma' \vdash^{\cap} \sigma}{\Gamma' \vdash^{\cap} M : \sigma}$$

- case [Intersect]: By inversion

$$\Gamma \vdash^{\cap} M : \sigma$$
$$\Gamma \vdash^{\cap} M : \sigma'$$
$$\sigma \cap \sigma' :: \tau$$

By IH

$$\Gamma' \vdash^{\cap} M : \sigma$$
$$\Gamma' \vdash^{\cap} M : \sigma'$$

So, the following derivation is valid

$$\textsc{Intersect}\ \frac{\Gamma' \vdash^{\cap} M : \sigma \qquad \Gamma' \vdash^{\cap} M : \sigma' \qquad \sigma \cap \sigma :: \tau}{\Gamma' \vdash^{\cap} M : \sigma \cap \sigma'}$$

- case [Var-B]: By inversion

$$\Gamma(y) = \sigma_1 \cap \cdots \cap \sigma_n$$
$$\sigma_i :: B(\forall i. 1 \leq i \leq n)$$

As $y \neq x$ we have $\Gamma(y) = \Gamma'(y)$. So, the following derivation is valid

$$\textsc{Var-B}\ \frac{\Gamma'(y) = \sigma_1 \cap \cdots \cap \sigma_n \qquad \sigma_i :: B(\forall i. 1 \leq 1 \leq n)}{\Gamma' \vdash^{\cap} y : \{\nu : B \mid \nu = y\}}$$

- case [Var]: Very similar to the previous case.

- case [App]: By inversion

$$\Gamma \vdash^{\cap} M : (y : \sigma' \to \sigma)$$
$$\Gamma \vdash^{\cap} N : \sigma'$$

By IH

$$\Gamma' \vdash^{\cap} M : (y : \sigma' \to \sigma)$$
$$\Gamma' \vdash^{\cap} N : \sigma'$$

So, the following derivation is valid

$$\textsc{App}\ \frac{\Gamma' \vdash^{\cap} M : (y : \sigma' \to \sigma) \qquad \Gamma' \vdash^{\cap} N : \sigma'}{\Gamma \vdash^{\cap} MN : [N/y]\sigma}$$

- case [Fun]: By inversion

$$\Gamma; y : \sigma \vdash^{\cap} M : \sigma'$$
$$\Gamma \vdash^{\cap} \sigma$$
$$\sigma :: \tau$$

By IH and (4.)

$$\Gamma'; y : \sigma \vdash^{\cap} M : \sigma'$$
$$\Gamma' \vdash^{\cap} \sigma$$

So, the following derivation is valid

$$\textsc{Fun}\ \frac{\Gamma'; y : \sigma \vdash^{\cap} M : \sigma' \qquad \Gamma' \vdash^{\cap} \sigma \qquad \sigma :: \tau}{\Gamma' \vdash^{\cap} \lambda y.M : \sigma \to \sigma'}$$

- case [Const]: Easy, by the following derivation

$$\frac{\textsc{Const}}{\Gamma' \vdash^{\cap} \mathrm{c} : ty(\mathrm{c})}$$

- case [Let]: By inversion

$$\Gamma \vdash^{\cap} M : \sigma'$$
$$\Gamma; y : \sigma' \vdash^{\cap} N : \sigma$$
$$\Gamma \vdash^{\cap} \sigma$$

By IH and (4.)

$$\Gamma' \vdash^{\cap} M : \sigma'$$
$$\Gamma'; y : \sigma' \vdash^{\cap} N : \sigma$$
$$\Gamma' \vdash^{\cap} \sigma$$

So, the following derivation is valid

$$\textsc{Let}\ \frac{\Gamma' \vdash^{\cap} M : \sigma' \qquad \Gamma'; y : \sigma' \vdash^{\cap} N : \sigma \qquad \Gamma' \vdash^{\cap} \sigma}{\Gamma' \vdash^{\cap} \text{let } y = M \text{ in } N : \sigma}$$

- case [Gen]: By inversion

$$\Gamma \vdash^{\cap} M : \sigma$$
$$\alpha \notin \Gamma$$

By IH

$$\Gamma' \vdash^{\cap} M : \sigma$$

We still have $\alpha \notin \Gamma'$, since Γ' only differs from Γ by the binding $x : \sigma_x$.
So, the following derivation is valid

$$\textsc{Gen}\ \frac{\Gamma' \vdash^{\cap} M : \sigma \qquad \alpha \notin \Gamma'}{\Gamma' \vdash^{\cap} [\Lambda\alpha]M : \forall\alpha.\sigma}$$

- case [Inst]: By inversion

$$\Gamma \vdash^{\cap} M : \forall\alpha.\sigma$$
$$\Gamma \vdash^{\cap} \sigma'$$
$$\text{Shape}(\sigma') = \tau$$

By IH and (4.)

$$\Gamma' \vdash^{\cap} M : \forall\alpha.\sigma$$
$$\Gamma' \vdash^{\cap} \sigma'$$

So, the following derivation is valid

$$\textsc{Inst}\ \frac{\Gamma' \vdash^{\cap} M : \forall\alpha.\sigma \qquad \Gamma' \vdash^{\cap} \sigma' \qquad \text{Shape}(\sigma') = \tau}{\Gamma' \vdash^{\cap} [\tau]M : [\sigma'/\alpha]\sigma}$$

□

Lemma 2 (Substitution)**.** *If*

$$\Gamma_1 \vdash^{\cap} v : \sigma'$$
$$\Gamma = \Gamma_1; x : \sigma'; \Gamma_2$$
$$\Gamma' = \Gamma_1; [v/x]\Gamma_2$$

then:

1. *if* $\Gamma \models \rho_1; [v/x]\rho_2$ *then* $\Gamma' \models \rho_1; \rho_2$*;*
2. *if* $\Gamma \vdash^{\cap} \phi \Rightarrow \phi'$ *then* $\Gamma' \vdash^{\cap} [v/x]\phi \Rightarrow [v/x]\phi'$*;*
3. *if* $\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2$ *then* $\Gamma' \vdash^{\cap} [v/x]\sigma_1 \prec [v/x]\sigma_2$*;*
4. *if* $\Gamma \vdash^{\cap} \sigma$ *then* $\Gamma' \vdash^{\cap} [v/x]\sigma$*;*
5. *if* $\sigma :: \tau$ *then* $[v/x]\sigma :: \tau$*;*
6. *if* $\Gamma \vdash^{\cap} M : \sigma$ *then* $\Gamma' \vdash^{\cap} M : \sigma$*.*

Proof. By simultaneous induction on the derivations.

1. We split by cases on the structure of Γ.

   - case $\Gamma = \emptyset$: Trivial, since Γ contains at least $x : \sigma'$
   - case $\Gamma = \Gamma_1; x : \sigma'; \Gamma_2'; y : \sigma''$: For this case

$$\Gamma_2 = \Gamma_2'; y : \sigma''$$
$$\rho_2 = \rho_2'; [v'/y]$$

   By inversion on rule [CS-Ext]:

$$\Gamma_1; x : \sigma'; \Gamma' \models \rho_1; [v/x]; \rho_2$$
$$\emptyset \vdash^{\cap} v' : \rho_1; [v/x]; \rho_2'(\sigma'')$$

   We have that $\text{FreeVars}(v') = \emptyset$, since it is typed with an empty environment. By IH and (5.)

$$\Gamma_1; [v/x]\Gamma_2' \models \rho_1; \rho_2'$$
$$\emptyset \vdash^{\cap} [v/x]v' : [v/x](\rho_1; [v/x]; \rho_2'(\sigma'')$$

   Since $\rho_1; [v/x]; \rho_2' = \rho_1; \rho_2'; [v/x]$, the following derivation is valid

$$\textsc{CS-Ext}\ \frac{\Gamma_1; [v/x]\Gamma_2' \models \rho_1; \rho_2' \qquad \emptyset \vdash^{\cap} [v/x]v' : [v/x](\rho_1; \rho_2'; \rho_2'(\sigma'')}{\Gamma_1; [v/x]\Gamma_2'; y : [v/x]\sigma'' \models \rho_1; \rho_2'; [[v/x]v'/y]}$$

Given that $[v/x]v' = v'$, the previous derivation could be written

$$\textsc{CS-Ext}\ \frac{\Gamma_1; [v/x]\Gamma_2' \models \rho_1; \rho_2' \qquad \emptyset \vdash^{\cap} [v/x]v' : [v/x](\rho_1; \rho_2'; \rho_2'(\sigma''))}{\Gamma_1; [v/x]\Gamma_2'; y : [v/x]\sigma'' \models \rho_1; \rho_2'; [v'/y]}$$

where $\Gamma_1; [v/x]\Gamma_2'; y : [v/x]\sigma'' = \Gamma_1; [v/x]\Gamma_2$ and $\rho_1; \rho_2'; [v'/y] = \rho_1; \rho_2$

2. By inversion on rule [Imp]

$$\begin{array}{l}\Gamma \vdash^{\cap} \phi : \mathit{bool} \\ \Gamma \vdash^{\cap} \phi' : \mathit{bool} \\ \forall \rho. (\Gamma \models \rho \text{ and } \rho(\phi) \overset{*}{\rightsquigarrow} \top \text{ implies} \rho(\phi') \overset{*}{\rightsquigarrow} \top\end{array}$$

By (5.)

$$\begin{array}{c}\Gamma' \vdash^{\cap} [v/x]\phi : \mathit{bool} \\ \Gamma \vdash^{\cap} [v/x]\phi' : \mathit{bool}\end{array}$$

By the form of Γ, ρ must be of the form $\rho_1; [v/x]; \rho_2$, so

$$\forall \rho_1, \rho_2. (\Gamma \models \rho_1; [v/x]; \rho_2 \text{ and } \rho_1; [v/x]; \rho_2(\phi) \overset{*}{\rightsquigarrow} \top \text{ implies} \rho_1; [v/x]; \rho_2(\phi') \overset{*}{\rightsquigarrow} \top$$

We have $\rho_1; [v/x]; \rho_2 = \rho_1; \rho_2; [v/x]$, so

$$\forall \rho_1, \rho_2. (\Gamma \models \rho_1; [v/x]; \rho_2 \text{ and } \rho_1; \rho_2; [v/x](\phi) \overset{*}{\rightsquigarrow} \top \text{ implies} \rho_1; \rho_2; [v/x](\phi') \overset{*}{\rightsquigarrow} \top$$

By (1.)

$$\forall \rho_1, \rho_2. (\Gamma' \models \rho_1; \rho_2 \text{ and } \rho_1; \rho_2; [v/x](\phi) \overset{*}{\rightsquigarrow} \top \text{ implies} \rho_1; \rho_2; [v/x](\phi') \overset{*}{\rightsquigarrow} \top$$

The following derivation is then valid

$$\textsc{Imp}\ \frac{\begin{array}{c}\Gamma' \vdash^{\cap} [v/x]\phi : \mathit{bool} \qquad \Gamma \vdash^{\cap} [v/x]\phi' : \mathit{bool} \\ \forall \rho_1, \rho_2. (\Gamma' \models \rho_1; \rho_2 \text{ and } \rho_1; \rho_2; [v/x](\phi) \overset{*}{\rightsquigarrow} \top \text{ implies} \rho_1; \rho_2; [v/x](\phi') \overset{*}{\rightsquigarrow} \top\end{array}}{\Gamma' \vdash^{\cap} [v/x]\phi \Rightarrow [v/x]\phi'}$$

3. By induction on the derivation of $\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2$, splitting cases on which rule was used at the bottom.

- case [$\prec$-Base]: By inversion

$$\Gamma; \nu : B \vdash^{\cap} \phi_1 \wedge \cdots \wedge \phi_n \Rightarrow \phi_1' \wedge \cdots \wedge \phi_m'$$

By IH (2.) and as $[v/x]B = B$

$$\Gamma'; \nu : B \vdash^{\cap} [v/x](\phi_1 \wedge \cdots \wedge \phi_n) \Rightarrow [v/x](\phi_1' \wedge \cdots \wedge \phi_m')$$

So, the following derivation is valid

$$\prec\text{-Base}\ \frac{\Gamma'; \nu : B \vdash^{\cap} [v/x](\phi_1 \wedge \cdots \wedge \phi_n) \Rightarrow [v/x](\phi_1' \wedge \cdots \wedge \phi_m')}{\Gamma' \vdash^{\cap} [v/x](\{\nu : B \mid \phi_1\} \cap \cdots \cap \{\nu : B \mid \phi_n\}) \prec [v/x](\{\nu : B \mid \phi_1'\} \cap \cdots \cap \{\nu : B \mid \phi_m'\})}$$

- case [$\prec$-Intersect-Fun]: since

$$[v/x](y : \sigma \to \sigma_1) \cap [v/x](y : \sigma \to \sigma_2) = (y : [v/x]\sigma \to [v/x]\sigma_1) \cap (y : [v/x]\sigma \to [v/x]\sigma_2)$$

the desired conclusion holds by the following derivation

$$\prec\text{-Intersect-Fun}\ \frac{}{\Gamma' \vdash^{\cap} [v/x](y : \sigma \to \sigma_1) \cap [v/x](y : \sigma \to \sigma_2) \prec [v/x](y : \sigma \to \sigma_1 \cap \sigma_2)}$$

- case [$\prec$-Fun]: By inversion
$$\Gamma \vdash^{\cap} \sigma_1' \prec \sigma_1$$
$$\Gamma; y : \sigma_1' \vdash^{\cap} \sigma_2 \prec \sigma_2'$$
By IH
$$\Gamma' \vdash^{\cap} [v/x]\sigma_1' \prec [v/x]\sigma_1$$
$$\Gamma'; [v/x]y : \sigma_1' \vdash^{\cap} [v/x]\sigma_2 \prec [v/x]\sigma_2'$$
By the definition of substitution, the following derivation is valid
$$\prec\text{-Fun}\ \frac{\Gamma' \vdash^{\cap} [v/x]\sigma_1' \prec [v/x]\sigma_1 \qquad \Gamma'; [v/x]y : \sigma_1' \vdash^{\cap} [v/x]\sigma_2 \prec [v/x]\sigma_2'}{\Gamma' \vdash^{\cap} y :\prec [v/x]\sigma_1 \to [v/x]\sigma_2 \prec y :\prec [v/x]\sigma_1' \to [v/x]\sigma_2'}$$

- case [$\prec$-Var]: Easy, by the following derivation
$$\prec\text{-Var}\ \frac{}{\Gamma' \vdash \alpha \prec \alpha}$$

- case [$\prec$-Left]: By the definition of substitution, the desired conclusion holds by the following derivation
$$\prec\text{-Left}\ \frac{}{\Gamma' \vdash [v/x]\sigma_1 \cap [v/x]\sigma_2 \prec [v/x]\sigma_1}$$

- case [$\prec$-Right]: By the definition of substitution, the desired conclusion holds by the following derivation
$$\prec\text{-Right}\ \frac{}{\Gamma' \vdash [v/x]\sigma_1 \cap [v/x]\sigma_2 \prec [v/x]\sigma_2}$$

- case [$\prec$-Intersect]: By inversion
$$\Gamma \vdash^{\cap} \sigma \prec \sigma_1$$
$$\Gamma \vdash^{\cap} \sigma \prec \sigma_2$$
By IH
$$\Gamma' \vdash^{\cap} [v/x]\sigma \prec [v/x]\sigma_1$$
$$\Gamma' \vdash^{\cap} [v/x]\sigma \prec [v/x]\sigma_2$$
By the definition of substitution, the desired conclusion holds by the following derivation
$$\prec\text{-Intersect}\ \frac{\Gamma' \vdash^{\cap} [v/x]\sigma \prec [v/x]\sigma_1 \qquad \Gamma' \vdash^{\cap} [v/x]\sigma \prec [v/x]\sigma_2}{\Gamma' \vdash^{\cap} [v/x]\sigma \prec [v/x]\sigma_1 \cap [v/x]\sigma_2}$$

- case [$\prec$-Poly]: By inversion
$$\Gamma \vdash^{\cap} \sigma_1 \prec \sigma_2$$
By IH
$$\Gamma' \vdash^{\cap} [v/x]\sigma_1 \prec [v/x]\sigma_2$$
By the definition of substitution, the desired conclusion holds by the following derivation
$$\prec\text{-Poly}\ \frac{\Gamma' \vdash^{\cap} [v/x]\sigma_1 \prec [v/x]\sigma_2}{\Gamma' \vdash^{\cap} \forall\alpha.[v/x]\sigma_1 \prec \forall\alpha.[v/x]\sigma_2}$$

4. By induction on the derivation of $\Gamma \vdash^{\cap} \sigma$, splitting cases on which rule was used at the bottom.

- case [WF-B]: By inversion
$$\Gamma; \nu : B \vdash^{\cap} \phi : \mathit{bool}$$
By IH (6.):
$$\Gamma'; \nu : [v/x]B \vdash^{\cap} \phi : \mathit{bool}$$
Since $[v/x]B = B$, the desired conclusion holds by the following derivation
$$\text{WF-B}\ \frac{\Gamma'; \nu : B \vdash^{\cap} \phi : \mathit{bool}}{\Gamma' \vdash^{\cap} \{\nu : B \mid \phi\}}$$

- case [WF-Var]: Easy, by the following derivation

$$\Gamma' \vdash^{\cap} \alpha$$

- case [WF-Fun]: By inversion

$$\Gamma; y : \sigma_1 \vdash^{\cap} \sigma_2$$

By IH

$$\Gamma'; y : [v/x]\sigma_1 \vdash^{\cap} [v/x]\sigma_2$$

By the definition of substitution, the desired conclusion holds by the following derivation

$$\text{WF-Fun}\ \frac{\Gamma'; y : [v/x]\sigma_1 \vdash^{\cap} [v/x]\sigma_2}{\Gamma' \vdash^{\cap} y : [v/x]\sigma_1 \to [v/x]\sigma_2}$$

- case [WF-Poly]: By inversion

$$\Gamma \vdash^{\cap} \sigma$$

By IH

$$\Gamma' \vdash^{\cap} [v/x]\sigma$$

By the definition of substitution, the desired conclusion holds by the following derivation

$$\text{WF-Poly}\ \frac{\Gamma' \vdash^{\cap} [v/x]\sigma}{\Gamma' \vdash^{\cap} \forall\alpha.[v/x]\sigma}$$

- case [WF-Intersect]: By inversion

$$\Gamma \vdash^{\cap} \sigma_1$$
$$\Gamma \vdash^{\cap} \sigma_2$$

By IH

$$\Gamma' \vdash^{\cap} [v/x]\sigma_1$$
$$\Gamma' \vdash^{\cap} [v/x]\sigma_2$$

By the definition of substitution, the desired conclusion holds by the following derivation

$$\text{WF-Intersect}\ \frac{\Gamma' \vdash^{\cap} [v/x]\sigma_1 \qquad \Gamma' \vdash^{\cap} [v/x]\sigma_2}{\Gamma' \vdash^{\cap} [v/x]\sigma_1 \cap [v/x]\sigma_2}$$

5. By induction on the derivation of $\sigma :: \tau$. This item clearly holds, since a substitution $[v/x]\sigma$ only affects refinement expressions, maintaining the correspondent ML type. So, the derivation for $\sigma :: \tau$ is the same for $[v/x]\sigma :: \tau$.

6. By induction on the derivation of $\Gamma \vdash^{\cap} M : \sigma$, splitting by cases on which rule is used at the bottom.

- case [Sub]: By inversion

$$\Gamma \vdash^{\cap} M : \sigma'$$
$$\Gamma \vdash^{\cap} \sigma' \prec \sigma$$
$$\Gamma \vdash^{\cap} \sigma$$

By IH, (3.) and (4.)

$$\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma'$$
$$\Gamma' \vdash^{\cap} [v/x]\sigma' \prec [v/x]\sigma$$
$$\Gamma' \vdash^{\cap} [v/x]\sigma$$

So, the following derivation is valid

$$\text{Sub}\ \frac{\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma' \qquad \Gamma' \vdash^{\cap} [v/x]\sigma' \prec [v/x]\sigma \qquad \Gamma' \vdash^{\cap} [v/x]\sigma}{\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma}$$

- case [Intersect]: By inversion

$$\Gamma \vdash^{\cap} M : \sigma$$
$$\Gamma \vdash^{\cap} M : \sigma'$$
$$\sigma \cap \sigma' :: \tau$$

[v/x]
By IH, and (5.)

$$\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma$$
$$\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma'$$
$$([v/x]\sigma \cap [v/x]\sigma') :: \tau$$

By the definition of substitution, the desired conclusion holds by the following derivation

$$\textsc{Intersect}\ \frac{\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma \qquad \Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma' \qquad ([v/x]\sigma \cap [v/x]\sigma') :: \tau}{\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma \cap [v/x]\sigma'}$$

- case [Var-B]: By inversion

$$\Gamma(y) = \sigma_1 \cap \cdots \cap \sigma_n, \quad \sigma_i = \{\nu : B \mid \phi_i\}$$
$$\sigma_i :: B(\forall i.1 \leq i \leq n)$$

Two sub-cases follow: either $y = x$ or $y \neq x$;

  - sub-case $y = x$: For this case we have $\sigma' = \{\nu : B \mid \phi_x\}$.
    By the definition of substitution
    $$[v/x]y = v$$
    So, v is a variable of basic type and we have $\Gamma_1(v) = \sigma''$.
    By Lemma (1)
    $$\Gamma_1; [v/x]\Gamma_2(y) = \sigma''$$
    The following derivation is valid
    $$\textsc{Var-B}\ \frac{\Gamma_1; [v/x]\Gamma_2(y) = \sigma'' \qquad \sigma'' :: B}{\Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} v : \{\nu : B \mid \nu = v\}}$$
    Given that $v = [v/x]y$ then we have
    $$\Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} v : \{\nu : B \mid \nu = v\} \equiv \Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} [v/x]y : [v/x]\{\nu : B \mid \nu = y\}$$
    which is precisely the desired conclusion.
  - sub-case $y \neq x$: For this $[v/x]y = y$, so
    $$[v/x]\{\nu : B \mid \nu = y\} = \{\nu : B \mid \nu = y\}$$
    We have
    $$\Gamma_1; [v/x]\Gamma_2(y) = \{\nu : B \mid [v/x]\phi_1\} \cap \cdots \cap \{\nu : B \mid [v/x]\phi_n\}$$
    $$\sigma_i :: B$$
    So, the following derivation is valid
    $$\Gamma_1$$

- case [Var]: By inversion

$$\Gamma(y) \text{ not a base type}$$
$$\Gamma(y) :: \tau$$

– sub-case $y = x$: For this case $[v/x]y = v$.
Given that $y = x$ then

$$\Gamma \vdash^{\cap} \sigma' \prec \sigma$$

By (3.)

$$\Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} [v/x]\sigma' \prec [v/x]\sigma$$

By Lemma 1

$$\Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} v : \sigma'$$

We have $x \notin \text{Freevars}(\sigma')$, so $[v/x]\sigma' = \sigma'$
By (4.) we have

$$\Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} [v/x]\sigma$$

So, the following derivation is valid

$$\textsc{Var}\ \frac{\Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} v : [v/x]\sigma' \quad \Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} [v/x]\sigma' \prec [v/x]\sigma \qquad \Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} [v/x]\sigma}{\Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} v : [v/x]\sigma}$$

which is the desired conclusion, since $[v/x]y = v$.

– sub-case $y \neq x$: For this case $[v/x]y = y$.
By inversion

$$\Gamma(y) = \sigma$$
$$\sigma :: \tau$$

If $y \in \text{dom}(\Gamma_1)$ then the result is immediate, since $[v/x]\sigma = \sigma$.
If $y \in \text{dom}(\Gamma_2)$, then $\Gamma_2(y) = \sigma$ and so it holds that $(\Gamma_1; [v/x]\Gamma_2)(y) = [v/x]\sigma$.
By (5.)

$$[v/x]\sigma :: \tau$$

The following derivation is then valid

$$\textsc{Var}\ \frac{(\Gamma_1; [v/x]Gamma_2)(y) = [v/x]\sigma \qquad [v/x]\sigma :: \tau}{\Gamma_1; [v/x]\Gamma_2 \vdash^{\cap} y : [v/x]\sigma}$$

which is the desired conclusion, since $[v/x]y = y$.

- case [App]: By inversion

$$\Gamma \vdash^{\cap} M : (y : \sigma' \to \sigma)$$
$$\Gamma \vdash^{\cap} N : \sigma'$$

By IH

$$\Gamma' \vdash^{\cap} [v/x]M : [v/x](y : \sigma' \to \sigma)\Gamma' \vdash^{\cap} [v/x]N : [v/x]\sigma'$$

By the definition of substitution

$$[v/x](y : \sigma' \to \sigma) = (y : [v/x]\sigma' \to [v/x]\sigma)$$

So, the following derivation is valid

$$\textsc{App}\ \frac{\Gamma' \vdash^{\cap} [v/x]M : [v/x](y : \sigma' \to \sigma) \qquad \Gamma' \vdash^{\cap} [v/x]N : [v/x]\sigma'}{\Gamma' \vdash^{\cap} MN : [N/y][v/x]\sigma}$$

As $[N/y][v/x]\sigma = [v/x]([N/y]\sigma)$ the desired conclusion follows by the previous derivation.

- case [Fun]: By inversion
$$\begin{array}{l}\Gamma; y : \sigma \vdash^{\cap} M : \sigma' \\ \Gamma \vdash^{\cap} \sigma \\ \sigma :: \tau\end{array}$$
By IH, (4.) and (5.)
$$\begin{array}{l}\Gamma'; y : [v/x]\sigma \vdash^{\cap} [v/x]M : [v/x]\sigma' \\ \Gamma' \vdash^{\cap} [v/x]\sigma \\ ([v/x]\sigma) :: \tau\end{array}$$
The following derivation is then valid
$$\textsc{Fun}\ \frac{\Gamma'; y : [v/x]\sigma \vdash^{\cap} [v/x]M : [v/x]\sigma' \qquad \Gamma' \vdash^{\cap} [v/x]\sigma \qquad ([v/x]\sigma) :: \tau}{\Gamma' \vdash^{\cap} \lambda y.M : y : [v/x]\sigma \to [v/x]\sigma'}$$
Since $y : [v/x]\sigma \to [v/x]\sigma' = [v/x](y : \sigma \to \sigma'$ then the desired conclusion follows by the previous derivation.
- case [Const]: easy, since $\text{FreeVars}(ty(\text{c})) = \emptyset$ and then
$$[v/x]ty(\text{c}) = ty(\text{c})$$
- case [Let]: By inversion
$$\begin{array}{l}\Gamma \vdash^{\cap} M : \sigma' \\ \Gamma; y : \sigma' \vdash^{\cap} N : \sigma \\ \Gamma \vdash^{\cap} \sigma\end{array}$$
By IH and (4.)
$$\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma'\Gamma'; y : [v/x]\sigma' \vdash^{\cap} [v/x]N : [v/x]\sigma\Gamma' \vdash^{\cap} [v/x]\sigma$$
So, the following derivation is valid
$$\textsc{Let}\ \frac{\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma' \qquad \Gamma'; y : [v/x]\sigma' \vdash^{\cap} [v/x]N : [v/x]\sigma \qquad \Gamma' \vdash^{\cap} [v/x]\sigma}{\Gamma' \vdash^{\cap} \text{let } y = [v/x]M \text{ in } [v/x]N : [v/x]\sigma}$$
- case [Gen]: By inversion
$$\begin{array}{l}\Gamma \vdash^{\cap} M : \sigma \\ \alpha \notin \Gamma\end{array}$$
By IH
$$\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma$$
The free variables of Γ' are the same of Γ (substitutions only affect refinement expressions), so
$$\alpha \notin \Gamma'$$
The following derivation is then valid
$$\textsc{Gen}\ \frac{\Gamma' \vdash^{\cap} [v/x]M : [v/x]\sigma \qquad \alpha \notin \Gamma'}{\Gamma' \vdash^{\cap} [v/x][\Lambda\alpha]M : [v/x]\forall\alpha.\sigma}$$
- case [Inst]: By inversion
$$\begin{array}{l}\Gamma \vdash^{\cap} M : \forall\alpha.\sigma \\ \Gamma \vdash^{\cap} \sigma' \\ \text{Shape}(\sigma') = \tau\end{array}$$
By IH and (4.)
$$\Gamma' \vdash^{\cap} [v/x]M : [v/x]\forall\alpha.\sigma\Gamma' \vdash^{\cap} [v/x]\sigma'$$

The following derivation is then valid

$$\textsc{Inst}\ \frac{\Gamma' \vdash^{\cap} [v/x]M : [v/x]\forall\alpha.\sigma \qquad \Gamma' \vdash^{\cap} [v/x]\sigma' \qquad \text{Shape}([v/x]\sigma') = \tau}{\Gamma' \vdash^{\cap} [v/x][\tau]M : [[v/x]\sigma'/\alpha][v/x]\sigma}$$

Since $[[v/x]\sigma'/\alpha] = [v/x][\sigma'/\alpha]$, the desired conclusion follows by the previous derivation.

□

Theorem 3 (Subject reduction)**.** *If* $\Gamma \vdash^{\cap} M : \sigma$ *and* $M \rightsquigarrow N$ *then* $\Gamma \vdash^{\cap} N : \sigma$*.*

Proof. By induction on the derivation $\Gamma \vdash^{\cap} M : \sigma$, splitting cases on which rule was used at the bottom.

- case [Sub]: By inversion
$$\Gamma \vdash^{\cap} M : \sigma'$$
$$\Gamma \vdash^{\cap} \sigma' \prec \sigma$$
$$\Gamma \vdash^{\cap} \sigma$$
By IH
$$\Gamma \vdash^{\cap} N : \sigma'$$
So, the following derivation is valid
$$\textsc{Sub}\ \frac{\Gamma \vdash^{\cap} N : \sigma' \qquad \Gamma \vdash^{\cap} \sigma' \prec \sigma \qquad \Gamma \vdash^{\cap} \sigma}{\Gamma \vdash^{\cap} N : \sigma}$$

- case [Intersect]: By inversion
$$\Gamma \vdash^{\cap} M : \sigma$$
$$\Gamma \vdash^{\cap} M : \sigma'$$
$$\sigma \cap \sigma' :: \tau$$
By IH
$$\Gamma \vdash^{\cap} N : \sigma \Gamma \vdash^{\cap} N : \sigma'$$
So, the following derivation is then valid
$$\textsc{Intersect}\ \frac{\Gamma \vdash^{\cap} N : \sigma \qquad \Gamma \vdash^{\cap} N : \sigma' \qquad \sigma \cap \sigma' :: \tau}{\Gamma \vdash^{\cap} N : \sigma \cap \sigma'}$$

- cases [Var-B], [Var], [Fun] and [Const]: Trivial, since these terms can't be further reduced.

- case [App]: By inversion
$$\Gamma \vdash^{\cap} M : (x : \sigma' \to \sigma)$$
$$\Gamma \vdash^{\cap} N : \sigma'$$
  - sub-case in which M is a context: For this case consider $M \rightsquigarrow M'$.
    By IH
    $$\Gamma \vdash^{\cap} M' : (x : \sigma' \to \sigma)$$
    Given that $M \rightsquigarrow M'$, then $MN \rightsquigarrow M'N$.
    The following derivation is then valid
    $$\textsc{App}\ \frac{\Gamma \vdash^{\cap} M' : (x : \sigma' \to \sigma) \qquad \Gamma \vdash^{\cap} N : \sigma'}{\Gamma \vdash^{\cap} M'N : [N/x]\sigma}$$
  - sub-case in which N is a context: Similar to the previous one.

– sub-case in which application is of the form $\mathrm{c}\,v$: By pushing applications of rule [Sub] down, we can ensure rule [Const] was used at the bottom of the derivation of the type for c.

For this case, $\mathrm{c}\,v \rightsquigarrow [\![\mathrm{c}]\!](v)$.

By inversion

$$\Gamma \vdash^{\cap} \mathrm{c} : (x : \sigma' \to \sigma)$$
$$\Gamma \vdash^{\cap} v : sigma'$$

By Definition 1, we have

$$\Gamma \vdash^{\cap} [\![\mathrm{c}]\!](v) : [v/x]\sigma$$

which is the desired conclusion.

– case in which application is of the form $(\lambda x.M)v$: For this case

$$(\lambda x.M)v \rightsquigarrow [v/x]M$$

By pushing applications of the rule [Sub] down, we can ensure rule [Fun] is used at the bottom of the derivation of the type for $\lambda x.M$.

By inversion

$$\Gamma \vdash^{\cap} \lambda x.M : (x : \sigma' \to \sigma)$$
$$\Gamma \vdash^{\cap} v : \sigma'$$

By inversion on rule [Fun]

$$x : \sigma' \vdash^{\cap} M : \sigma$$

By Lemma 2

$$\Gamma \vdash^{\cap} [v/x]M : [v/x]\sigma$$

which is the desired conclusion.

- case [Let]:

  – sub-case in which M is a value: For this case

  $$\text{let } x = v \text{ in } N \rightsquigarrow [v/x]N$$

  By inversion

  $$\Gamma \vdash^{\cap} v : \sigma'$$
  $$\Gamma; x : \sigma' \vdash^{\cap} N : \sigma$$
  $$\Gamma \vdash^{\cap} \sigma$$

  Since $\Gamma \vdash^{\cap} \sigma$, $x \notin \text{FreeVars}(\sigma)$, so

  $$[v/x]\sigma = \sigma$$

  By Lemma 2

  $$\Gamma \vdash^{\cap} [v/x]N : [v/x]\sigma$$

  which is the desired conclusion.

  – sub-case in which M is a context: For this cas

  $$\text{let } x = M \text{ in } N \rightsquigarrow \text{let } x = M' \text{ in } N$$

  By inversion

  $$\Gamma \vdash^{\cap} M : \sigma'$$
  $$\Gamma; x : \sigma' \vdash^{\cap} N : \sigma$$
  $$\Gamma \vdash^{\cap} \sigma$$

  By IH

  $$\Gamma \vdash^{\cap} M' : \sigma'$$

So, the following derivation is valid

$$\textsc{Let}\ \frac{\Gamma \vdash^{\cap} M' : \sigma' \Gamma; x : \sigma' \vdash^{\cap} N : \sigma \qquad \Gamma \vdash^{\cap} \sigma}{\text{let } x = M' \text{ in } N : \sigma}$$

- case [Gen]: By inversion

$$\begin{array}{l} \Gamma \vdash^{\cap} M : \sigma \\ \alpha \notin \Gamma \end{array}$$

By IH

$$\Gamma \vdash^{\cap} N : \sigma$$

So, the following derivation is valid

$$\textsc{Gen}\ \frac{\Gamma \vdash^{\cap} N : \sigma \qquad \alpha \notin \Gamma}{\Gamma \vdash^{\cap} [\Lambda\alpha]N : \forall\alpha.\sigma}$$

- case [Inst]: By inversion

$$\begin{array}{l} \Gamma \vdash^{\cap} M : \forall\alpha.\sigma \\ \Gamma \vdash^{\cap} \sigma' \\ \text{Shape}(\sigma') = \tau \end{array}$$

By IH

$$\Gamma \vdash^{\cap} N : \forall\alpha.\sigma$$

So, the following derivation is valid

$$\textsc{Inst}\ \frac{\Gamma \vdash^{\cap} N : \forall\alpha.\sigma \qquad \Gamma \vdash^{\cap} \sigma' \qquad \text{Shape}(\sigma') = \tau}{\Gamma \vdash^{\cap} [\tau]N : [\sigma'/\alpha]\sigma}$$

□

Theorem 4 (Over approximation)**.** *If* $\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma$ *then* $\Gamma \vdash^{\cap} M : \sigma$.

Proof. The proof follows by straightforward induction on the typing derivation. At each case the key observation is that each Liquid Intersection Type is also a Dependent Intersection Type and for each rule in the decidable system there is a matching rule on the undecidable side. For the case of [$\prec$-Base] we use Definition 1. □

B Soundness of type inference

Theorem 5 (Soundness)**.** *if* $\texttt{Infer}(\Gamma, M, \mathbb{Q}) = \sigma$ *then* $\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma$

Proof. By structural induction over M.

- case $M \equiv x$:
  - subcase in which M has a basic type in this case $\mathcal{W}(\text{Shape}(\Gamma), x) = B$ and so x has type $\{v : B \mid \phi_1\} \cap \ldots \cap \{v : B \mid \phi_n\}$, which we abbreviate to $\sigma_1 \cap \cdots \cap \sigma_n$.
    The following derivation is then valid

$$\frac{\Gamma(x) = \sigma_1 \cap \cdots \cap \sigma_n \qquad \sigma_i :: B(\forall i.1 \leq i \leq n)}{\Gamma \vdash^{\cap}_{\mathbb{Q}} x : \{v = x\}}\ \text{B-Var}$$

– subcase in which x has not a basic type: in this case $\sigma = \Gamma(x)$.
So, the following derivation is valid

$$\frac{\Gamma(x) = \sigma \qquad \Gamma(x) :: \tau}{\Gamma \vdash^{\cap}_{\mathbb{Q}} x : \sigma}\ \textsc{Var}$$

- Case $M \equiv c$: Easy, by application of the rule [Const].
- Case $M \equiv \lambda x.N$: In this case the algorithm computes
  - $(x : \sigma_1 \to \tau_1) \cap \ldots \cap (x : \sigma_n \to \tau_n) = \text{Fresh}(\mathcal{W}(\text{Shape}(\Gamma), \lambda x.M), \mathbb{Q})$
  - the type $\mathcal{A}$ of τ'_i such that $\tau'_i = \text{Cons}(\Gamma; x : \sigma_i, N, \mathbb{Q})$

  By IH

$$\Gamma; x : \sigma_i \vdash^{\cap}_{\mathbb{Q}} N : \tau'_i \tag{a}$$

  The type $\mathcal{A}$ restricts the type only to the well formed intersections: $\Gamma \vdash^{\cap} (x : \sigma_1 \to \tau_1) \cap \ldots \cap (x : \sigma_n \to \tau_n)$ translates to $\{\Gamma \vdash^{\cap} (x : \sigma_1 \to \tau_1), \ldots, (x : \sigma_n \to \tau_n)\}$
  Consider the sub-set of derivations in (a) such that $\Gamma; x : \sigma_j \vdash^{\cap}_{\mathbb{Q}} \tau'_j \lhd \tau_j$ and that respect the type $\mathcal{A}$.
  We have then a set of derivations of the form

$$\textsc{Sub}\ \frac{\dfrac{\Gamma; x : \sigma_j \vdash^{\cap}_{\mathbb{Q}} N : \tau'_j \qquad \Gamma; x : \sigma_j \vdash^{\cap}_{\mathbb{Q}} \tau'_j \lhd \tau_j \qquad \Gamma; x : \sigma_j \vdash^{\cap} \tau_j}{\Gamma; x : \sigma_j \vdash^{\cap}_{\mathbb{Q}} N : \tau_j} \qquad \Gamma \vdash^{\cap} x : \sigma_j \to \tau_j}{\Gamma \vdash^{\cap}_{\mathbb{Q}} \lambda x.N : (x : \sigma_j \to \tau_j)}\ \textsc{Abs}$$

  By repeated application of the rule [Intersect]

$$\frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} \lambda x.N : (x : \sigma_j \to \tau_j) \qquad \ldots \qquad \Gamma \vdash^{\cap}_{\mathbb{Q}} \lambda x.N : (x : \sigma_{j+k} \to \tau_{j+k})}{\Gamma \vdash^{\cap}_{\mathbb{Q}} \lambda x.N : (x : \sigma_j \to \tau_j) \cap \ldots \cap (x : \sigma_{j+k} \to \tau_{j+k})}\ \textsc{Intersect}$$

- case $M \equiv M'N$: By IH
  - $\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : (x : \sigma_1 \to \tau_1) \cap \ldots \cap (x : \sigma_n \to \tau_n)$
  - $\Gamma \vdash^{\cap}_{\mathbb{Q}} N : \sigma$

  For all the σ_i such that $\sigma \lhd \sigma_i$ we have a derivation of the form

$$\textsc{Sub}\ \frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : (x : \sigma_1 \to \tau_1) \cap \ldots \cap (x : \sigma_n \to \tau_n) \quad \Gamma \vdash^{\cap}_{\mathbb{Q}} (x : \sigma_1 \to \tau_1) \cap \ldots \cap (x : \sigma_n \to \tau_n) \lhd (x : \sigma_i \to \tau_i)}{\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : (x : \sigma_i \to \tau_i)} \qquad \mathcal{D}}{\Gamma \vdash^{\cap}_{\mathbb{Q}} M'N : \tau_i[N/x]}\ \textsc{App}$$

  in which $\mathcal{D}$ is

$$\frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} N : \sigma \qquad \Gamma \vdash^{\cap}_{\mathbb{Q}} \sigma \lhd \sigma_i \qquad \Gamma \vdash^{\cap} \sigma_i}{\Gamma \vdash^{\cap}_{\mathbb{Q}} N : \sigma_i}\ \textsc{Sub}$$

  Let $\mathcal{D}_1$ be the entire previous derivation.
  For each σ_i that satisfy $\sigma \lhd \sigma_i$ we have a derivation of the previous form.
  So, by repeated application of the rule [Intersect] the following derivation is valid

$$\frac{\mathcal{D}_i \qquad \ldots \qquad \mathcal{D}_{i+j}}{\Gamma \vdash^{\cap}_{\mathbb{Q}} M'N : \tau_i[N/x] \cap \ldots \cap \tau_{i+j}[N/x]}\ \textsc{Intersect}$$

  By the definition of substitution we have $\tau_i[N/x] \cap \ldots \cap \tau_{i+j}[N/x] = (\tau_i \cap \ldots \cap \tau_{i+j})[N/x]$, which is precisely the inferred type.

- case $M \equiv \text{let}\, x = M' \,\text{in}\, N$:

  σ is of the form $\sigma'_1 \cap \ldots \cap \sigma'_n$.

  By IH

  - $\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : \sigma_1$
  - $\Gamma; x : \sigma_1 \vdash^{\cap}_{\mathbb{Q}} N : \sigma_2$

  The type $\mathcal{A}$ stands for the set of τ_i such that $\Gamma \vdash^{\cap} \tau_i$, which by the definition of well formed type we have

$$\textsc{WF-Intersect}\ \frac{\Gamma \vdash^{\cap} \sigma'_i \qquad \Gamma \vdash^{\cap} \sigma'_{i+j}}{\Gamma \vdash^{\cap} \sigma'_i \cap \ldots \cap \sigma'_{i+j}} \tag{b}$$

  Consider all σ''_i in $\mathcal{A}$ such that $\Gamma; x : \sigma_1 \vdash^{\cap}_{\mathbb{Q}} \sigma_2 \lhd \sigma''_i$.

  The following derivation is valid

$$\textsc{Let}\ \frac{\mathcal{D} \qquad \overset{(c)}{\dfrac{\textsc{Sub}\ \dfrac{\Gamma; x : \sigma_1 \vdash^{\cap}_{\mathbb{Q}} N : \sigma_2 \qquad \Gamma; x : \sigma_1 \vdash^{\cap}_{\mathbb{Q}} \sigma_2 \lhd \sigma''_i}{\Gamma; x : \sigma_1 \vdash^{\cap}_{\mathbb{Q}} N : \sigma''_i} \qquad \ldots \qquad \Gamma; x : \sigma_1 \vdash^{\cap}_{\mathbb{Q}} N : \sigma''_{i+j}}{\Gamma; x : \sigma_1 \vdash^{\cap}_{\mathbb{Q}} N : \sigma''_i \cap \ldots \cap \sigma''_{i+j}}\ \textsc{Intersect}}{\Gamma \vdash^{\cap}_{\mathbb{Q}} \sigma''_i \cap \ldots \cap \sigma''_{i+j}}}{\Gamma \vdash^{\cap}_{\mathbb{Q}} \text{let}\, x = M \,\text{in}\, N : \sigma''_i \cap \ldots \cap \sigma''_{i+j}}$$

  in which $\mathcal{D}$ is

$$\Gamma \vdash^{\cap}_{\mathbb{Q}} M : \sigma_1$$

  The derivation (c) is valid by (b), since $\sigma''_i \cap \ldots \cap \sigma''_{i+j}$ is a sub-type of $\sigma'_i \cap \ldots \cap \sigma'_{i+j}$.

- case $M \equiv [\Lambda\alpha]M'$:

  By IH

$$\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : \sigma$$

  The following derivation is valid

$$\frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : \sigma \qquad \alpha \notin \Gamma}{\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : \forall\alpha.\sigma}\ \textsc{Gen}$$

- case $M \equiv [\tau]M'$:

  By IH

$$\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : \forall\alpha.\sigma$$

  Since $\tau' = \text{Fresh}(\tau, \mathbb{Q})$, then $\tau = \text{Shape}(\tau')$.

  τ' is of the form $\tau'_1 \cap \ldots \cap \tau'_n$.

  The type $\mathcal{A}$ stands for the set of all τ'_i such that $\Gamma \vdash^{\cap} \tau'_i$, so a sub-type of $\tau'_1 \cap \ldots \cap \tau'_n$.

  Then, the following derivation is valid

$$\frac{\Gamma \vdash^{\cap}_{\mathbb{Q}} M' : \forall\alpha.\sigma \qquad \dfrac{\textsc{Intersect}\ \dfrac{\Gamma \vdash^{\cap} \tau'_i \qquad \ldots \qquad \Gamma \vdash^{\cap} \tau'_{i+j}}{\Gamma \vdash^{\cap} \tau'_i \cap \ldots \cap \tau'_{i+j}} \qquad \text{Shape}(\tau'_i \cap \ldots \cap \tau'_{i+j}) = \tau}{\Gamma \vdash^{\cap}_{\mathbb{Q}} [\tau]M' : \sigma[\tau'_i \cap \ldots \cap \tau'_{i+j}/\alpha]}\ \textsc{Inst}$$

□