Mode checking in the Concurrent Logical Framework

Jorge Luis Sacchini* **Iliano Cervesato***
Frank Pfenning† **Carsten Schürmann‡**

August 2014
CMU-CS-14-134
CMU-CS-QTR-123

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

†Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
*Carnegie Mellon University, Qatar campus
‡IT University of Copenhagen — Copenhagen, Denmark

The authors can be reached at sacchini@qatar.cmu.edu, iliano@cmu.edu, fp@cs.cmu.edu, and carsten@itu.dk.

Abstract

We define and prove correct a mode checker for a significant fragment of the concurrent logical framework CLF.

* This paper was made possible by grant 09-1107-1-168, *Formal Reasoning about Languages for Distributed Computation*, from the Qatar National Research Fund (a member of the Qatar Foundation). The statements made herein are solely the responsibility of the authors.

Keywords: Mode Checking, Logical Frameworks, Concurrent Logical Framework

Contents

List of Figures

1 Introduction

The Edinburgh Logical Framework (LF) [5] is a logical framework based on a simple dependently-typed theory. It is designed to specify and reason about programming languages and deductive systems. One particular feature of LF that makes it suitable for this task is the use of higher-order abstract syntax (HOAS) for specifying object variables as LF variables. Therefore, properties of variables and contexts (such as substitution and weakening) in the object language are obtained "for free", as they map directly to the same property in LF.

The LF specification of a system (e.g., a programming language or a logic) is given by a set of datatypes describing the object language and a set of relations describing the system semantics. For example, the operational semantics of a programming language can be defined as a relation $\mathsf{eval}\, E\, V$ between a program E and a value V, meaning that E evaluates to V. This semantics can be executed as in logic programming, using backward chaining and unification.

A relation often defines a function (e.g., the eval relation above defines a function from programs to values). Such functional reading can be specified through a *mode declaration* that defines arguments as *inputs* or *outputs*. Mode declarations are used in systems like Twelf [10] as part of the totality checker. They can also be used to give specialized operational semantics that rely on matching instead of full unification. To ensure a mode declaration is valid, it is necessary to check that all rules describing the relation respect the declared modes.

In this report, we define mode checking for an extension of LF called the Concurrent Logical Framework (CLF) [4, 18]. CLF has been designed to specify and reason about concurrent and distributed programming languages. It extends LF with linear and affine types, as well as monadic types based on lax logic. The semantics of CLF combines backward chaining (as in Twelf) with committed-choice forward chaining [7].

The main contribution of this report is the definition of a mode checker for a substantial subset of CLF which we call Mini-CLF. To show the correctness of the mode checker, we define a mode-aware semantics, i.e., a semantics that uses mode declarations to compute the output arguments of a functional relation, given ground values for the inputs. This semantics fails if, during search, the input arguments are not ground. For well-moded programs, we show that this cannot happen. That is, during proof search, the input arguments in a goal are ground, and solving the goal grounds the output arguments. Furthermore, the mode-aware semantics is complete, in the sense that if there is a derivation for a well-moded goal, the semantics will find it.

Related work CLF was introduced by Watkins et al. in [4, 18]. The semantics of CLF is based on the semantics of Lollimon described in [7]. The semantics combines backward chaining as in LF with forward chaining for the monadic fragment. In [7], two criteria for stopping forward chaining are defined: saturation (when taking a further step does not change the current context) and quiescence (when no further steps can be taken). For the purpose of mode checking, we do not need to commit to any stopping criterion or clause-selection mechanism, as the mode checker should be valid in all cases.

Mode checking is used in Twelf in combination with termination and coverage checking to ensure that a sequence of declarations represent a valid proof [10]. See [11] for a description of mode checking in LF; our development is closer to that of Sarnat [13].

Rest of the report The presentation is organized in two stages. In Sect. 2 we define and prove correct a mode-checker for standard LF. In Sect. 3, we extend LF with linear and monadic types to obtain Mini-CLF, a substantial subset of CLF. The mode-aware semantics, mode checker, and the proof of correctness for Mini-CLF are obtained as extensions to the definitions and proofs given in Sect. 2. In Appendix A we define the mode checker for the full language of CLF.

2 Mode checking in LF

In this section we define and prove correct a mode checker for LF. In Sect. 2.1 we define the syntax and operational semantics of LF. As we explain below, we use a different presentation of LF that is easier to extend to CLF. In Sect. 2.2 we define the mode-checker for LF. The proof of correctness is given in Sect. 2.3.

2.1 Syntax and semantics of LF

We use a non-standard definition of LF that makes explicit certain aspects of LF related to the operational semantics and mode checking, as explained below. For completeness, we first give the standard presentation of LF.

Standard LF LF is a basic dependently-typed lambda calculus. Its syntax is defined as follows:

$$
\begin{array}{rr}
K ::= \mathsf{type} \mid \Pi x{:}A.K & \text{(Kinds)} \\
A ::= a \cdot S \mid \Pi x{:}A_1.A_2 & \text{(Types)} \\
S ::= () \mid N; S & \text{(Spines)} \\
N ::= \lambda x.N \mid H \cdot S & \text{(Terms)} \\
H ::= x \mid c & \text{(Heads)} \\
\Gamma ::= \cdot \mid \Gamma, x{:}A & \text{(Contexts)} \\
\Sigma ::= \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A & \text{(Signatures)}
\end{array}
$$

Terms are classified by types, which are themselves classified by kinds. We use the spine notation [1] for defining terms and types, so application is defined as a head (either a variable or constant) applied to a spine (a sequence of terms). Contexts are sequences of declarations; we write $\#\Gamma$ for the number of declarations in context Γ.

We make use of the usual metatheoretical properties of LF including existence of canonical forms, hereditary substitutions, weakening, strengthening, etc. (see, e.g., [6] for proofs of these properties).

Alternative presentation of LF We use a non-standard presentation of LF that can be easily extended with features from CLF (cf. Sect. 3). The alternative syntax of LF is the following:

$$
\begin{array}{rr}
K ::= \Pi\Gamma.\mathsf{type} & \text{(Kinds)} \\
A^- ::= \Pi\Gamma.A^+ \to P & \text{(Negative types)} \\
A^+ ::= \mathbf{1} \mid A^- \times A^+ & \text{(Positive types)} \\
P ::= a{\cdot}S & \text{(Atomic types)} \\
S ::= () \mid N; S & \text{(Spines)} \\
\Gamma ::= \cdot \mid \Gamma, x{:}A^- & \text{(Contexts)} \\
\Sigma ::= \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A^- & \text{(Signatures)}
\end{array}
$$

As in the polarized definition of CLF [15], we divide types between negative and positive types. LF types in the standard presentation correspond to negative types.

We make explicit the distinction between non-dependent and dependent product types. Dependent arguments are joined in context (without loss of generality, we assume that dependent arguments occur in front). Non-dependent arguments are uncurried in a product type (associated to the right).

This distinction between dependent and non-dependent arguments stems from the operational semantics: dependent arguments are solved by matching (or unification), while non-dependent arguments represent goals to be solved by proof search.

We leave the syntax of terms implicit, as it will not play a major role in the following. One consequence of this assumption is that the operational semantics does not construct a proof term, but only computes the substitution for the free variables. However, extending the semantics to compute proof terms is straightforward.

Typing rules They are given by the judgments:

- $\Gamma \vdash K : \mathsf{kind}$ meaning that kind K is well-typed in context Γ;
- $\Gamma \vdash A^- : \mathsf{type}$ (resp. $\Gamma \vdash A^+ : \mathsf{type}$) meaning that type A^- (resp. type A^+) is well-typed in context Γ;
- $\Gamma \vdash S : \Gamma' > \mathsf{type}$ meaning that spine S is well-typed for a kind of type $\Pi\Gamma'.\mathsf{type}$ in context Γ;
- $\Gamma \vdash N : A^-$ meaning that term N has type A^- in context Γ (this judgment is left unspecified);
- $\Gamma \vdash \Gamma'$ meaning that context Γ' is well-typed in context Γ.
- $\vdash \Sigma$ meaning that signature Σ is well typed.

We assume a fixed global signature Σ that is left implicit in the rules. The typing rules are given in Fig. 2.1. In the rules for spines, we use hereditary substitutions to compute the canonical form of substituting x by N in Γ' (denoted $[N/x]\Gamma'$). See [6] for a definition of hereditary substitutions for LF.

$\boxed{\Gamma \vdash K : \mathsf{kind}}$

$$\frac{\Gamma \vdash \Gamma'}{\Gamma \vdash \Pi\Gamma'.\mathsf{type} : \mathsf{kind}}$$

$\boxed{\Gamma \vdash A^- : \mathsf{type}}$

$$\frac{\Gamma \vdash \Gamma' \qquad \Gamma, \Gamma' \vdash A^+ : \mathsf{type} \qquad a{:}\Pi\Gamma_0.\mathsf{type} \in \Sigma \qquad \Gamma, \Gamma' \vdash S : \Gamma_0 > \mathsf{type}}{\Gamma \vdash \Pi\Gamma'.A^+ \to a \cdot S : \mathsf{type}}$$

$\boxed{\Gamma \vdash A^+ : \mathsf{type}}$

$$\frac{}{\Gamma \vdash \mathbf{1} : \mathsf{type}} \qquad \frac{\Gamma \vdash A^- : \mathsf{type} \qquad \Gamma \vdash B^+ : \mathsf{type}}{\Gamma \vdash A^- \times B^+ : \mathsf{type}}$$

$\boxed{\Gamma \vdash S : \Gamma' > \mathsf{type}}$

$$\frac{}{\Gamma \vdash () : \cdot > \mathsf{type}} \qquad \frac{\Gamma \vdash N : A^- \qquad \Gamma \vdash S : [N/x]\Gamma' > \mathsf{type}}{\Gamma \vdash N; S : (x{:}A^-), \Gamma' > \mathsf{type}}$$

$\boxed{\Gamma \vdash \Gamma'}$

$$\frac{}{\Gamma \vdash \cdot} \qquad \frac{\Gamma \vdash \Gamma' \qquad \Gamma, \Gamma' \vdash A^- : \mathsf{type}}{\Gamma \vdash \Gamma', x{:}A^-}$$

Figure 2.1: Typing rules of LF

$\boxed{\Gamma \vdash \rho : \Phi'}$

$$\frac{}{\Phi \vdash \cdot : \cdot} \qquad \frac{\Phi' \vdash \rho : \Phi \qquad |\Phi'| \vdash \rho A^- : \mathsf{type}}{\Phi', \forall^? x{:}\rho A^- \vdash \rho, \forall^? x / x : \Phi, \forall^? x{:}A^-}$$

$$\frac{\Phi' \vdash \rho : \Phi \qquad |\Phi'| \vdash N : \rho A^-}{\Phi' \vdash \rho, \exists N / x : \Phi, \exists x{:}A^-}$$

Figure 2.2: Typing rules for generalized substitutions

Preliminary definitions We introduce some notions needed to define the operational semantics. Following [13, 11] we consider an operational semantics based on *mixed-prefix contexts* to differentiate between existential variables that are solved by unification, and universal variables that are treated as constants (eigenvariables). Using mixed-prefix contexts avoids introducing logic variables. Mixed-prefix contexts are defined as follows:

$$\Phi ::= \cdot \mid \Phi, \underline{\forall} x{:}A^- \mid \Phi, \forall x{:}A^- \mid \Phi, \exists x{:}A^-$$

We distinguish between universal clauses ($\underline{\forall}$) and universal parameters ($\forall$); universal clauses can be used during search while universal parameters are only used during matching (or unification). This means that dependent arguments are introduced in the context using $\forall$, while non-dependent arguments are introduced by $\underline{\forall}$ (see rule $\mathsf{sem\text{-}type}^-$). We write $\forall^? x$ for a generic declaration that is either $\underline{\forall} x$ or $\forall x$. Existential variables are dependent variables that will be substituted by actual terms through unification (or matching).

We write $|\cdot|$ for the erasure function that transforms a mixed-prefix context into a regular context by erasing all quantifiers. It it defined by the following rules:

$$|\cdot| = \cdot \qquad |\Phi, \exists x{:}A^-| = |\Phi|, x{:}A^-$$
$$|\Phi, \underline{\forall} x{:}A^-| = |\Phi|, x{:}A^- \qquad |\Phi, \forall x{:}A^-| = |\Phi|, x{:}A^-$$

Given a context Γ, we write $\underline{\forall}\Gamma$ (resp. $\forall\Gamma$, $\exists\Gamma$) for the mixed-prefix context obtained by prefixing each declaration in Γ with $\underline{\forall}$ (resp. $\forall$, $\exists$).

A *generalized substitution* is a substitution that applies to the existential variables of a mixed-prefix context. They are defined as follows:

$$\rho ::= \cdot \mid \rho, \underline{\forall} x / x \mid \rho, \forall x / x \mid \rho, \exists N / x$$

Applying a substitution to a term (resp. a spine) is denoted ρN (resp. ρS). The result is the canonical form obtained by performing the hereditary substitution of ρ in N (resp. S).

A generalized substitution is well typed from Φ to Φ', denoted $\Phi' \vdash \rho : \Phi$, if it satisfies the rules given in Fig. 2.2. We write id_Φ for the identity substitution for context Φ that replaces every existential variable in Φ with its η-expansion; formally it is defined by the following equations:

$$\begin{aligned} \mathrm{id}_\cdot &= \cdot \\ \mathrm{id}_{\Phi, \forall^? x{:}A^-} &= \mathrm{id}_\Phi, \forall^? x / x \\ \mathrm{id}_{\Phi, \exists x{:}A^-} &= \mathrm{id}_\Phi, \mathsf{expand}_{\mathrm{id}_\Phi A^-}(x) \end{aligned}$$

It is easy to check that $\Phi \vdash \mathrm{id}_\Phi : \Phi$ for any context Φ.

Given a substitution ρ we write $\rho|_\Phi$ for the restriction of ρ to the domain of Φ. The following lemmas state properties about restriction. They are proved by induction on the typing derivation. In the first lemma, when restriction cuts out an existential context, the typing context Φ' remains the same. However, in the second lemma, when cutting out a universal context, we also need to change the typing context Φ'.

Lemma 2.1 *If* $\Phi' \vdash \rho : \Phi, \exists\Gamma$, *then* $\Phi' \vdash \rho|_\Phi : \Phi$.

Lemma 2.2 *If* $\Phi', \underline{\forall}\Gamma' \vdash \rho : \Phi, \underline{\forall}\Gamma$, *with* $\#\Gamma = \#\Gamma'$, *then* $\Phi' \vdash \rho|_\Phi : \Phi$.

A term N is ground with respect to a mixed-prefix context Φ, denoted $\Phi \vdash N\ \mathsf{grd}$, if N does not contain any existentially quantified variable from Φ. A term N (resp. a spine S) is in the *pattern fragment* with respect to Φ, denoted $\Phi \vdash N\ \mathsf{pat}$ (resp. $\Phi \vdash S\ \mathsf{pat}$), if every existential variable in N (resp. in S) is applied to a pattern (i.e., to distinct universal variables). We write $\Phi \vdash_x N\ \mathsf{pat}$ (resp. $\Phi \vdash_x S\ \mathsf{pat}$) to mean that the existentially quantified variable x occurs in N (resp. in S) as the head of a term applied to a pattern.

The following lemma states that substitutions preserve groundness.

Lemma 2.3 *If* $\Phi \vdash S\ \mathsf{grd}$ *and* $\Phi' \vdash \rho : \Phi$, *then* $\Phi' \vdash \rho S\ \mathsf{grd}$.

Operational Semantics The operational semantics of LF is defined as in logic programming: given a type A^- (a goal) in a mixed-prefix context Φ, find a substitution $\Phi' \vdash \rho : \Phi$ and a term N such that $|\Phi'| \vdash N : \rho A^-$ and $|\Phi'| \vdash N\ \mathsf{grd}$. Since we do not specify the syntax of terms, our semantics only computes the substitution ρ.

The operational semantics is defined by the judgments

$$\Phi \vdash A^- \Longrightarrow \rho; \Phi'$$
$$\Phi \vdash A^+ \Longrightarrow \rho; \Phi'$$

meaning that searching for a proof of A^- (resp. of A^+) in Φ returns a substitution ρ and a context Φ' such that $\Phi' \vdash \rho : \Phi$. We define a mode-aware semantics, which means that type families have a mode declaration that indicates input and output arguments. A mode is either input $(+)$ or output $(-)$. A mode declaration for a type family a is a sequence of modes whose length is the number of arguments of a; we denote it with $\mathsf{mode}(a)$. We assume that input arguments precede output arguments; we write $a \cdot \check{S}\ \hat{S}$ to separate the input $(\check{S})$ and output $(\hat{S})$ arguments.

We define the semantics based on *matching*, as opposed to using full unification as it is usual. This is possible because we consider a semantics that is mode aware. The matching algorithm is given by a judgment of the form

$$\Phi \vdash S_2 =: S_1 \Longrightarrow \square$$

meaning that matching S_1 against S_2 under context Φ gives a result $\square$. This result can be either a pair $(\rho; \Phi')$ where ρ is a substitution satisfying $\Phi' \vdash \rho : \Phi$ and $\rho S_1 = S_2$; or $\bot$ if the algorithm fails (meaning that no such substitution exists). A precondition is that S_1 is well typed in $|\Phi|$ and S_2 is ground. We say that S_1 is on the *pattern side* and S_2 is on the *value side* of the matching equation.

We do not define the matching algorithm explicitly, but only state the required properties as assumptions. We require the matching algorithm to be able to solve the *pattern fragment* [8], which is known to be decidable. This is stated in the following assumptions, which is used in the proofs in Sect. 2.3.

Assumption 2.4 *Given a context Φ and spines S_1 and S_2 such that $\Phi \vdash S_2$ grd and $\Phi \vdash S_1$ pat, then exactly one of the following holds:*

- $\Phi \vdash S_2 =: S_1 \Longrightarrow \bot$ *and there is no substitution ρ such that $\rho S_1 = S_2$;*
- *there exists a unique ρ such that $\rho S_1 = S_2$ and $\Phi \vdash S_2 =: S_1 \Longrightarrow \rho; \Phi'$ for some Φ'.*

The rules of the semantics are given in Fig. 2.3. Rule $\mathsf{sem\text{-}type}^-$ describe the execution of negative types. Dependent and non-dependent arguments are introduced as assumptions in the context. Dependent arguments are introduced as parameters, meaning they are not used for searching (see rule $\mathsf{sem\text{-}atm}$), while non-dependent arguments can be used for searching. We use the judgment $\vec{x}{:}A^- \Longrightarrow \underline{\forall}\Gamma_0$ to construct a context from a sequence of variables and a positive type; it is defined by the following rules:

$$\frac{}{\cdot{:}\mathbf{1} \Longrightarrow \cdot} \qquad \frac{\vec{x}{:}B^+ \Longrightarrow \underline{\forall}\Gamma}{x\,\vec{x}{:}A^- \times B^+ \Longrightarrow \underline{\forall}x{:}A^-, \underline{\forall}\Gamma}$$

Execution is then reduced to solving an atomic goal, which is handled by rule $\mathsf{sem\text{-}atm}$. In this case, we pick a clause c from the signature or the context which must have the same head as the goal we are trying to solve. Note that this clause must not be a parameter (i.e., it is not annotated with $\forall$). It then proceeds by matching the inputs ($\check{S}$ against $\check{S}_0$), solving the goals A_0^+, and matching the outputs ($\hat{S}_0$ against $\hat{S}$). The final result combines the substitutions obtained in each step. We do not specify any procedure for picking clauses, since the mode checker should be correct no matter which search procedure is used. Rule $\mathsf{sem\text{-}type}^+$ executes a sequence of goals from left to right, collecting the results.

Note that this semantics describes only successful queries. Queries that fail because the type is not inhabited or that lead to non-termination are represented by the absence of a derivation. We will extend this semantics to account for partial derivations in Sect. 2.3.1.

It is easy to check that derivations in the semantics compute substitutions between the input and output context.

Lemma 2.5 *If $\Phi \vdash A^- \Longrightarrow \rho; \Phi'$ then $\Phi' \vdash \rho : \Phi$.*

Proof: By mutual induction on the derivation using the equivalent statement for positive types, using Lemmas 2.1 and 2.2. □

2.2 Mode checking

The purpose of mode checking is to statically ensure that mode declarations will be respected at run-time (when running a query). This means that, every time the matching algorithm is invoked, the preconditions given in Assumption 2.4 are met, i.e., the value side is ground and the pattern side is in the pattern fragment.

$\boxed{\Phi \vdash A \Longrightarrow \rho; \Phi'}$

$$\frac{\vec{x}{:}A_0^+ \Longrightarrow \underline{\forall}\Gamma_0 \qquad \vec{x} \text{ fresh} \qquad \Phi, \forall\Gamma, \forall^?\Gamma_0 \vdash a{\cdot}S \Longrightarrow \rho; \Phi', \forall^?\Gamma_1 \qquad \#\Gamma_1 = \#(\Gamma, \Gamma_0)}{\Phi \vdash \Pi\Gamma.A^+ \to a{\cdot}S \Longrightarrow \rho|_\Phi; \Phi'}\ \textsf{sem-type}^-$$

$$\frac{\underline{\forall}c{:}\Pi\Gamma_0.A_0^+ \to a{\cdot}\check{S}_0\,\hat{S}_0 \in \Sigma, \Phi \qquad \Phi, \exists\Gamma_0 \vdash \check{S} =: \check{S}_0 \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1 \vdash \rho_1 A_0^+ \Longrightarrow \rho_2; \Phi_2 \qquad \Phi_2 \vdash \rho_2\rho_1\hat{S}_0 =: \rho_2\rho_1\hat{S} \Longrightarrow \rho_3; \Phi_3}{\Phi \vdash a{\cdot}\check{S}\,\hat{S} \Longrightarrow \rho_3\rho_2\rho_1|_\Phi; \Phi_3}\ \textsf{sem-atm}$$

$$\frac{\Phi \vdash A^- \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1 \vdash \rho_1 B^+ \Longrightarrow \rho_2; \Phi_2}{\Phi \vdash A^- \times B^+ \Longrightarrow \rho_2\rho_1; \Phi_2}\ \textsf{sem-type}^+$$

$$\frac{}{\Phi \vdash \mathbf{1} \Longrightarrow \mathrm{id}_\Phi; \Phi}\ \textsf{sem-one}$$

Figure 2.3: Semantics of LF

The mode checker can be seen as an abstract interpretation of a program. The result of executing the mode checker is an *abstract substitution* [11, 13] representing the groundness information of a generalized substitution. Abstract substitutions are defined by the following grammar:

$$\nu ::= \cdot \mid \nu, \forall x{:}A^-, \mid \nu, \exists^{\mathsf{u}} x{:}A^- \mid \nu, \exists^{\mathsf{g}} x{:}A^-$$

There is an obvious mapping from abstract substitutions to mixed-prefix contexts and to contexts. We denote them with $\downarrow\cdot$ and $|\cdot|$, respectively. Concretely, these mapping are defined by the following rules:

$$\begin{aligned} \downarrow(\cdot) &= \cdot & |\cdot| &= \cdot \\ \downarrow(\nu, \underline{\forall}x{:}A^-) &= \downarrow\nu, \underline{\forall}x{:}A^- & |\nu, \underline{\forall}x{:}A^-| &= |\nu|, x{:}A^- \\ \downarrow(\nu, \exists^{\mathsf{u}} x{:}A^-) &= \downarrow\nu, \exists x{:}A^- & |\nu, \exists^{\mathsf{u}} x{:}A^-| &= |\nu|, x{:}A^- \\ \downarrow(\nu, \exists^{\mathsf{g}} x{:}A^-) &= \downarrow\nu, \exists x{:}A^- & |\nu, \exists^{\mathsf{g}} x{:}A^-| &= |\nu|, x{:}A^- \end{aligned}$$

Given a mixed-prefix context Φ we denote with $\uparrow\Phi$ the abstract substitution obtained by changing every $\exists$ into $\exists^{\mathsf{u}}$:

$$\begin{aligned} \uparrow(\cdot) &= \cdot \\ \uparrow(\Phi, \underline{\forall}x{:}A^-) &= \uparrow\Phi, \underline{\forall}x{:}A^- \\ \uparrow(\Phi, \forall x{:}A^-) &= \uparrow\Phi, \underline{\forall}x{:}A^- \\ \uparrow(\Phi, \exists x{:}A^-) &= \uparrow\Phi, \exists^{\mathsf{u}} x{:}A^- \end{aligned}$$

We write $\mathrm{dom}^{\mathsf{g}}(\nu)$ (resp. $\mathrm{dom}^{\mathsf{u}}(\nu)$, $\mathrm{dom}^{\forall}(\nu)$) for the subset of variables declared in ν annotated with $\exists^{\mathsf{g}}$ (resp. $\exists^{\mathsf{u}}$, $\forall$).

Abstract substitutions define an abstraction over generalized substitutions, as expressed by the judgments $\Phi \vdash \rho : \nu$ and $\nu \vdash \rho : \nu'$ given by the rules of Fig. 2.4. Every existential ground variable in ν (declared with $\exists^{\mathsf{g}}$) must be substituted with a ground term (in Φ) by ρ. Unknown existential

$\boxed{\Phi \vdash \rho : \nu}$

$$\frac{}{\Phi \vdash \cdot : \cdot}$$

$$\frac{\Phi \vdash \rho : \nu \qquad |\Phi| \vdash \rho A^- : \mathsf{type}}{\Phi \vdash \rho, \forall^? x/x : \nu, \forall x{:}A^-}$$

$$\frac{\Phi \vdash \rho : \nu \qquad |\Phi| \vdash N : \rho A^- \qquad \Phi \vdash N \ \mathsf{grd}}{\Phi \vdash \rho, \exists N/x : \nu, \exists^{\mathsf{g}} x{:}A^-}$$

$$\frac{\Phi \vdash \rho : \nu \qquad |\Phi| \vdash N : \rho A^-}{\Phi \vdash \rho, \exists N/x : \nu, \exists^{\mathsf{u}} x{:}A^-}$$

$\boxed{\nu \vdash \rho : \nu'}$

$$\frac{}{\nu \vdash \cdot : \cdot}$$

$$\frac{\nu \vdash \rho : \nu \qquad |\nu| \vdash \rho A^- : \mathsf{type}}{\nu \vdash \rho, \forall^? x/x : \nu, \forall x{:}A^-}$$

$$\frac{\nu \vdash \rho : \nu \qquad |\nu| \vdash N : \rho A^- \qquad \nu \vdash N \ \mathsf{grd}}{\nu \vdash \rho, \exists N/x : \nu, \exists^{\mathsf{g}} x{:}A^-}$$

$$\frac{\nu \vdash \rho : \nu \qquad |\nu| \vdash N : \rho A^-}{\nu \vdash \rho, \exists N/x : \nu, \exists^{\mathsf{u}} x{:}A^-}$$

Figure 2.4: Typing rules for abstract substitutions

variables (declared with $\exists^{\mathsf{u}}$) can be substituted with any term, not necessarily ground. We write $\nu \vdash N$ grd to mean that N does not contain any existential unknown variable (declared with $\exists^{\mathsf{u}}$).

Given a context Φ and a substitution ρ there is usually more than one ν that satisfies $\Phi \vdash \rho : \nu$ (i.e., ρ can be abstracted in more than one way). This is captured by defining an order relation between abstract substitutions: $\nu_1 \sqsubseteq \nu_2$ is defined as the point-wise extension of the smallest order relation between declarations containing the rule

$$\exists^{\mathsf{g}} x{:}A^- \sqsubseteq \exists^{\mathsf{u}} x{:}A^-$$

Abstract substitutions obey the properties given next. The proofs proceed by induction on the relevant structure. The proofs are easy so we omit them.

Lemma 2.6 *If $\Phi \vdash \rho : \nu_1$ and $\nu_1 \sqsubseteq \nu_2$, then $\Phi \vdash \rho : \nu_2$.*

Lemma 2.7 *If $\Phi \vdash \rho : \nu, \exists^{?}\Gamma$, then $\Phi \vdash \rho|_\nu : \nu$.*

Lemma 2.8 *If $\Phi, \forall^{?}\rho\Gamma \vdash \rho : \nu, \forall\Gamma$, then $\Phi \vdash \rho|_\nu : \nu$.*

The mode checker It is defined as a static analysis performed on the program and the goal. It is defined by the following judgments:

- $\nu \vdash A^-$ $\mathsf{bwd} > \nu'$ checks that A^- is a valid declaration to be used in backward chaining, returning the output groundness information ν'.
- $\nu \vdash A$ $\mathsf{goal} > \nu'$ checks that A (either positive, negative, or atomic type) is a valid goal, returning the output groundness information ν'.
- $\nu \vdash A^-$ decl checks that A^- is a valid declaration. In LF, this judgment reduces to checking that A^- is valid for backward chaining ($\nu \vdash A^-$ $\mathsf{bwd} > \nu'$). However, in CLF, this depends on whether A^- is used for backward or forward chaining (cf. Sect. 3.2).
- $\nu \vdash S > \nu'$ infers groundness for all variables in S. If $\nu \vdash_x N$ pat with N a term in S and $\exists^{?}x{:}A^- \in \nu$, then $\exists^{\mathsf{g}}x{:}A^- \in \nu'$.

  We use the operation $\nu|^{\mathsf{g}}_{\mathcal{X}}$ for the abstract context obtained by grounding in ν the variables in $\mathcal{X}$; formally, it is defined by the following rules:

$$\cdot|^{\mathsf{g}}_{\mathcal{X}} = \cdot$$

$$\nu, \forall x{:}A^-|^{\mathsf{g}}_{\mathcal{X}} = \nu|^{\mathsf{g}}_{\mathcal{X}}, \forall x{:}A^-$$

$$\nu, \exists^{?}x{:}A^-|^{\mathsf{g}}_{\mathcal{X}} = \begin{cases} \nu|^{\mathsf{g}}_{\mathcal{X}}, \exists^{\mathsf{g}}x{:}A^- & \text{if } x \in \mathcal{X} \\ \nu|^{\mathsf{g}}_{\mathcal{X}}, \exists^{?}x{:}A^- & \text{otherwise} \end{cases}$$

- $\nu \vdash_{\mathsf{grd}} S$ checks that S is ground with respect to ν (i.e., S does not contain unknown existential variables).
- $\vdash \Sigma$ decl and $\nu \vdash \Phi$ decl check that all declarations in Σ and Φ are well moded.

The rules of the mode checker are given in Fig. 2.5. In rule mc-bwd we start by assuming that all dependent variables are unknown. We infer groundness information from the input arguments obtaining ν'. Then we infer groundness information from executing the goals in A^+, checking that solving goals would ground Γ. Finally, we check that the output arguments are ground.

In rule mc-goal, we infer groundness information from P and check that all types in A^+ are valid declarations. For atomic goals (rule mcg-atm) we check that input arguments are ground and infer groundness information for output arguments. Rules mcg-prod and mcg-one infer groundness information from all components of a positive type.

Rule mcd-bwd check that a declaration is valid for use in backward chaining. In the case of CLF, we will extend this judgment to account for declarations that can be used in forward chaining. The rest of the rules for declarations just iterate over the corresponding structure.

We prove some simple properties about the mode checking judgments, including substitution, weakening, and strengthening.

In the following, we write $\nu \vdash X > \nu'$ to mean a generic judgment that can be either of the form $\nu \vdash A^-$ bwd $> \nu'$, $\nu \vdash A$ goal $> \nu'$, or $\nu \vdash S > \nu'$. We have the following properties of the mode checker. We omit the easy proofs by induction on the relevant mode-checking derivation.

Lemma 2.9 *If $\nu \vdash X > \nu'$, then $\nu' \sqsubseteq \nu$.*

Lemma 2.10 *Let $\nu_1 \vdash X > \nu_2$ and $\nu_1' \sqsubseteq \nu_1$. Then, there exists $\nu_2' \sqsubseteq \nu_2$ such that $\nu_1' \vdash X > \nu_2'$.*

Lemma 2.11 (Weakening for mode checking) *Let $\nu_1, \nu_1' \vdash X > \nu_2, \nu_2'$ where $dom(\nu_1) = dom(\nu_2)$ and $dom(\nu_1') = dom(\nu_2')$. Then, $\nu_1, \nu_0, \nu_1' \vdash X > \nu_2, \nu_0, \nu_2'$.*

Lemma 2.12 (Strengthening for mode checking)

- *Let $\nu_1, \nu_0, \nu_1' \vdash X > \nu_2$, where $FV(X) \cap dom(\nu_0) = \emptyset$. Then there exists ν_2', ν_2'' such that $\nu_2 = \nu_2', \nu_0, \nu_2''$, $dom(\nu_1) = dom(\nu_2')$, $dom(\nu_1') = dom(\nu_2'')$, and $\nu_1, \nu_1' \vdash X > \nu_2', \nu_2''$.*
- *Let $\nu_1, \nu_0, \nu_1' \vdash_{\mathsf{grd}} A^-$ where $FV(A^-) \cap dom(\nu_0) = \emptyset$. Then $\nu_1, \nu_1' \vdash_{\mathsf{grd}} A^-$*

The following lemmas are key in showing that the mode checker is a form of abstract interpretation.

Lemma 2.13 (Substitution lemma for mode checking goals) *If $\nu_1 \vdash A$ goal $> \nu_2$ and $\nu_1' \vdash \rho : \nu_1$, then there exists ν_2' such that $\nu_1' \vdash \rho A$ goal $> \nu_2'$ and $\nu_2' \vdash \rho : \nu_2$.*

Lemma 2.14 (Substitution lemma for mode checking declarations)

1. *If $\nu \vdash A^-$ decl and $\nu' \vdash \rho : \nu$, then $\nu' \vdash \rho A$ decl.*
2. *If $\nu \vdash \Phi$ decl and $\nu' \vdash \rho : \nu$, then $\nu' \vdash \rho \Phi$ decl.*

2.3 Correctness of the mode checker

In this section we show that the mode checker is correct, by showing that mode checking is a form of abstract interpretation over the semantics.

We need the following technical lemma on the matching algorithm, which follows from Assumption 2.4.

$\boxed{\nu \vdash A^- \text{ bwd} > \nu'}$

$$\frac{\nu, \exists^{\mathsf{u}}\Gamma \vdash \check{S} > \nu' \qquad \nu' \vdash A^+ \text{ goal} > \nu'', \exists^{\mathsf{g}}\Gamma \qquad \nu'', \exists^{\mathsf{g}}\Gamma \vdash_{\mathsf{grd}} \hat{S}}{\nu \vdash \Pi\Gamma.A^+ \to a{\cdot}\check{S}\,\hat{S} \text{ bwd} > \nu''} \text{ mc-bwd}$$

$\boxed{\nu \vdash A \text{ goal} > \nu'}$

$$\frac{\nu, \forall\Gamma \vdash P \text{ goal} > \nu' \qquad \nu, \forall\Gamma \vdash A^+ \text{ decl}}{\nu \vdash \Pi\Gamma.A^+ \to P \text{ goal} > \nu'} \text{ mc-goal} \qquad \frac{\nu \vdash_{\mathsf{grd}} \check{S} \qquad \nu \vdash \hat{S} > \nu'}{\nu \vdash a{\cdot}\check{S}\,\hat{S} \text{ goal} > \nu'} \text{ mcg-atm}$$

$$\frac{\nu \vdash A^- \text{ goal} > \nu' \qquad \nu' \vdash B^+ \text{ goal} > \nu''}{\nu \vdash A^- \times B^+ \text{ goal} > \nu''} \text{ mcg-prod} \qquad \frac{}{\nu \vdash \mathbf{1} \text{ goal} > \nu} \text{ mcg-one}$$

$\boxed{\nu \vdash A \text{ decl}}$

$$\frac{\nu \vdash \Pi\Gamma.A^+ \to a{\cdot}\check{S}\,\hat{S} \text{ bwd} > \nu'}{\nu \vdash \Pi\Gamma.A^+ \to a{\cdot}\check{S}\,\hat{S} \text{ decl}} \text{ mcd-bwd}$$

$$\frac{\nu \vdash A^- \text{ decl} \qquad \nu \vdash B^+ \text{ decl}}{\nu \vdash A^- \times B^+ \text{ decl}} \text{ mcd-prod} \qquad \frac{}{\nu \vdash \mathbf{1} \text{ decl}} \text{ mcd-one}$$

$\boxed{\vdash \Sigma \text{ decl}}$

$$\frac{}{\vdash \cdot \text{ decl}} \text{ mcs-emp} \qquad \frac{\vdash \Sigma \text{ decl} \qquad \cdot \vdash A^- \text{ decl}}{\vdash \Sigma, a{:}A^- \text{ decl}} \text{ mcs-cons}$$

$\boxed{\nu \vdash \Phi \text{ decl}}$

$$\frac{}{\nu \vdash \cdot \text{ decl}} \text{ mcc-emp} \qquad \frac{\nu \vdash \Phi \text{ decl} \qquad \nu, {\uparrow}\Phi \vdash A^- \text{ decl}}{\nu \vdash \Phi, \underline{\forall}x{:}A^- \text{ decl}} \text{ mcc-}\forall$$

$$\frac{\nu \vdash \Phi \text{ decl}}{\nu \vdash \Phi, \forall x{:}A^- \text{ decl}} \text{ mcc-}\forall \qquad \frac{\nu \vdash \Phi \text{ decl}}{\nu \vdash \Phi, \exists^{?}x{:}A^- \text{ decl}} \text{ mcc-}\exists$$

$\boxed{\nu \vdash S > \nu'}$

$$\frac{\mathcal{X} = \{x : \nu \vdash_x S \text{ pat}\}}{\nu \vdash S > \nu|^{\mathsf{g}}_{\mathcal{X}}} \text{ mcsp-infer}$$

$\boxed{\nu \vdash_{\mathsf{grd}} S}$

$$\frac{\mathrm{FV}(S) \subseteq \mathrm{dom}^{\forall}(\nu) \cup \mathrm{dom}^{\mathsf{g}}(\nu)}{\nu \vdash_{\mathsf{grd}} S} \text{ mcsp-check}$$

Figure 2.5: Mode checker for LF

Lemma 2.15 *If* $\Phi \vdash S_2 =: S_1 \Longrightarrow \rho; \Phi'$, $\uparrow\Phi \vdash_{\mathsf{grd}} S_2$, *and* $\uparrow\Phi \vdash S_1 > \nu$, *then* $\Phi' \vdash \rho : \nu$.

The following theorem states that mode checking is a form of abstract interpretation. Concretely, it states that substitutions obtained from executing a query respect the groundness information obtained from mode checking. For example, consider a goal of the form $\exists\Gamma \vdash a{\cdot}\check{S}\,\hat{S}$, where $\check{S}$ is ground, and $\mathrm{FV}(\hat{S}) = \mathrm{dom}(\Gamma)$. If there exists a derivation $\exists^{\mathsf{u}}\Gamma \vdash a{\cdot}\check{S}\,\hat{S}$ $\mathsf{goal} > \exists^{\mathsf{g}}\Gamma$, then every substitution obtained from executing this goal grounds all variables in Γ (i.e. in $\hat{S}$).

Theorem 2.16 *Assume that* $\vdash \Sigma$ decl *and* $\cdot \vdash \Phi$ decl. *If* $\uparrow\Phi \vdash A$ $\mathsf{goal} > \nu$, *where* A *is either a positive, negative, or atomic type, and* $\Phi \vdash A \Longrightarrow \rho; \Phi'$, *then* $\cdot \vdash \Phi'$ decl *and* $\Phi' \vdash \rho : \nu$.

Proof: We proceed by induction on the derivation of the semantics and inversion on the derivation of the mode checker.

sem-type$^-$ We have the following derivation:

$$\dfrac{\begin{array}{c}\vec{x}{:}A_0^+ \Longrightarrow \underline{\forall}\Gamma_0 \qquad \vec{x} \text{ fresh} \\ \Phi, \forall\Gamma, \underline{\forall}\Gamma_0 \vdash a{\cdot}S \Longrightarrow \rho; \Phi', \forall^?\Gamma_1 \qquad \#\Gamma_1 = \#(\Gamma, \Gamma_0)\end{array}}{\Phi \vdash \Pi\Gamma.A^+ \to a{\cdot}S \Longrightarrow \rho|_\Phi; \Phi'}\ \mathsf{sem\text{-}type}^-$$

By inversion on $\uparrow\Phi \vdash \Pi\Gamma.A^+ \to P$ $\mathsf{goal} > \nu$ we have $\uparrow\Phi \vdash P$ $\mathsf{goal} > \nu$ and $\uparrow\Phi, \forall\Gamma \vdash A^+$ decl. From the latter and $\vec{x}{:}A_0^+ \Longrightarrow \underline{\forall}\Gamma_0$, we have $\cdot \vdash \uparrow\Phi, \forall\Gamma, \forall\Gamma_0$ decl. By IH, $\Phi', \forall^?\Gamma_1 \vdash \rho : \nu, \forall\Gamma, \Gamma_0$. The result follows by Lemma 2.8 (note that $\rho|_\nu$ is the same as $\rho|_\Phi$).

sem-atm We have the following derivation:

$$\dfrac{\begin{array}{c}\underline{\forall}c{:}\Pi\Gamma_0.A_0^+ \to a{\cdot}\check{S}_0\,\hat{S}_0 \in \Sigma, \Phi \\ \Phi, \exists\Gamma_0 \vdash \check{S} =: \check{S}_0 \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1 \vdash \rho_1 A_0^+ \Longrightarrow \rho_2; \Phi_2 \qquad \Phi_2 \vdash \rho_2\rho_1\hat{S}_0 =: \rho_2\rho_1\hat{S} \Longrightarrow \rho_3; \Phi_3\end{array}}{\Phi \vdash a{\cdot}\check{S}\,\hat{S} \Longrightarrow \rho_3\rho_2\rho_1|_\Phi; \Phi_3}$$

By inversion on the mode-checking derivation of $\uparrow\Phi \vdash a{\cdot}\check{S}\,\hat{S}$ $\mathsf{goal} > \nu$, we have $\nu \vdash_{\mathsf{grd}} \check{S}$ and $\uparrow\Phi \vdash \hat{S} > \nu$.

By inversion on the mode-checking derivation of the type of h and weakening (Lemma 2.11), we have $\uparrow\Phi \vdash \Pi\Gamma_0.A_0^+ \to a{\cdot}\check{S}_0\,\hat{S}_0$ $\mathsf{bwd} > \nu_0$, for some ν_0. Again by inversion, $\uparrow\Phi, \exists^{\mathsf{u}}\Gamma_0 \vdash \check{S}_0 > \nu_0'$, $\nu_0' \vdash A_0^+$ $\mathsf{goal} > \nu_0, \exists^{\mathsf{g}}\Gamma$, and $\nu_0, \exists^{\mathsf{g}}\Gamma \vdash_{\mathsf{grd}} \hat{S}_0$.

By Lemma 2.15 on the matching equation for inputs, we have $\Phi_1 \vdash \rho_1 : \nu_0'$. By Lemma 2.13, there exists ν_1 such that $\uparrow\Phi_1 \vdash \rho_1 A_0^+$ $\mathsf{goal} > \nu_1$ and $\nu_1 \vdash \rho_1 : \nu_0, \exists^{\mathsf{g}}\Gamma$.

By IH, we have $\Phi_2 \vdash \rho_2 : \nu_1$.

From $\uparrow\Phi \vdash \hat{S} > \nu$, by weakening, $\uparrow\Phi, \exists^{\mathsf{u}}\Gamma_0 \vdash \hat{S} > \nu, \exists^{\mathsf{u}}\Gamma_0$. Since $\nu_0' \sqsubseteq \uparrow\Phi, \exists^{\mathsf{u}}\Gamma_0$, there exists ν^* such that $\nu_0' \vdash \hat{S} > \nu^*$ and $\nu^* \sqsubseteq \nu, \exists^{\mathsf{u}}\Gamma_0$.

There exists ν^{**} such that $\Phi_2 \vdash \rho_2\rho_1\hat{S} > \nu^{**}$ and $\nu^{**} \vdash \rho_2\rho_1 : \nu^*$.

By Lemma 2.15 on the matching equation for outputs, we have $\Phi_3 \vdash \rho_3\rho_2\rho_1 : \nu^*$. From $\nu^* \sqsubseteq \nu, \exists^{\mathsf{u}}\Gamma_0$, we have $\Phi_3 \vdash \rho_3\rho_2\rho_1 : \nu, \exists^{\mathsf{u}}\Gamma_0$. The result follows by restricting the $\rho_3\rho_2\rho_1$ to Φ (same as restricting to ν).

sem-type$^+$ We have the following derivation:

$$\frac{\Phi \vdash A^- \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1 \vdash \rho_1 B^+ \Longrightarrow \rho_2; \Phi_2}{\Phi \vdash A^- \times B^+ \Longrightarrow \rho_2\rho_1; \Phi_2}$$

By inversion on $\uparrow\Phi \vdash A^- \times B^+ \ \mathsf{goal} > \nu$, there exists ν' such that $\uparrow\Phi \vdash A^- \ \mathsf{goal} > \nu'$ and $\nu' \vdash B^+ \ \mathsf{goal} > \nu$. By IH on A^- we have $\Phi_1 \vdash \rho_1 : \nu'$. By Lemma 2.13, there exists ν'' such that $\uparrow\Phi_1 \vdash \rho_1 B^+ \ \mathsf{goal} > \nu''$ and $\nu'' \vdash \rho_1 : \nu$. By IH on B^+ we have $\Phi_2 \vdash \rho_2 : \nu''$. By composing ρ_2 and ρ_1 we have the desired result $\Phi_2 \vdash \rho_2\rho_1 : \nu$.

sem-one Follows immediately.

□

A consequence of this lemma is that if a program is well moded, the semantics respects mode declarations (in a sense, well-moded programs cannot go wrong).

Definition 2.17 *A derivation $\Phi \vdash A \Longrightarrow \rho; \Phi'$ is mode correct if the following conditions hold:*

1. *every subderivation $\Phi_0 \vdash A_0 \Longrightarrow \rho_0; \Phi_0'$ satisfies $\cdot \vdash \Phi_0 \ \mathsf{decl}$ and $\uparrow\Phi_0 \vdash A_0 \ \mathsf{goal} > \nu$ for some ν;*
2. *every subderivation $\Phi_0 \vdash S_2 =: S_1 \Longrightarrow \rho_0; \Phi_0'$ (resp. $\Phi_0 \vdash S_2 =: S_1 \Longrightarrow \rho_0; \Phi_0'$) satisfies $\uparrow\Phi_0 \vdash_{\mathsf{grd}} S_2$ and $\uparrow\Phi_0 \vdash S_1 > \nu$ for some ν.*

Lemma 2.18 *Assume that $\vdash \Sigma \ \mathsf{decl}$, $\cdot \vdash \uparrow\Phi \ \mathsf{decl}$ and $\uparrow\Phi \vdash A \ \mathsf{goal} > \nu$ for some ν (where A is a positive, negative, or atomic type). Then any derivation $\Phi \vdash A \Longrightarrow \rho; \Phi'$ is mode correct.*

Proof: By induction on the semantics derivation and case analysis on the last rule used. Clause 1 of Def. 2.17 follows immediately for each rule. Clause 2 follows the same pattern as in Lemma 2.16. □

The following corollary is a particular and common case of Theorem 2.16. It shows that when solving goals of the form $a{\cdot}\check{S}\hat{S}$, with $\check{S}$ ground, the resulting substitution ground all variables in $\hat{S}$.

Corollary 2.19 *Assume that $\vdash \Sigma \ \mathsf{decl}$ and $\cdot \vdash \Phi \ \mathsf{decl}$. If $\exists^{\mathsf{u}}\Gamma \vdash a{\cdot}\check{S}\hat{S} \ \mathsf{goal} > \exists^{\mathsf{g}}\Gamma$, and $\Phi \vdash a{\cdot}\check{S}\hat{S} \Longrightarrow \rho; \cdot$, then $\cdot \vdash \rho : \exists^{\mathsf{g}}\Gamma$.*

2.3.1 Partial semantics

The previous lemma shows that, during the execution of successful queries, every call to the matching algorithm satisfies the conditions of Assumption 2.4. However, it does not say anything in the case of failing queries, since there is no derivation. To remedy this situation, we extend the semantics with partial executions, i.e., partial derivations, where execution is stopped at the point where the matching algorithm is called.

Formally, we extend the semantics with the judgment

$$\Phi \vdash^? A \rightharpoonup \rho; \Phi'$$

$\boxed{\Phi \vdash^? A \rightharpoonup \rho; \Phi'}$

$$\frac{\vec{x} : A_0^+ \Longrightarrow \underline{\forall}\Gamma_1 \qquad \Phi, \forall\Gamma_0, \underline{\forall}\Gamma_1 \vdash^? a{\cdot}S \rightharpoonup \rho; \Phi'}{\Phi \vdash^? \Pi\Gamma_0.A_0^+ \to a{\cdot}S \rightharpoonup \rho; \Phi'}\ \textsf{psem-type}^-$$

$$\frac{\underline{\forall}c{:}\Pi\Gamma_0.A_0^+ \to a{\cdot}\check{S}_0\,\hat{S}_0 \in \Sigma, \Phi}{\Phi \vdash^? a{\cdot}\check{S}\,\hat{S} \rightharpoonup \cdot; \Phi}\ \textsf{psem-atm-in}$$

$$\frac{\underline{\forall}c{:}\Pi\Gamma_0.A_0^+ \to a{\cdot}\check{S}_0\,\hat{S}_0 \in \Sigma, \Phi \qquad \Phi, \exists\Gamma_0 \vdash \check{S} =: \check{S}_0 \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1 \vdash \rho_1 A_0^+ \Longrightarrow \rho_2; \Phi_2}{\Phi \vdash^? a{\cdot}\check{S}\,\hat{S} \rightharpoonup \rho_2\rho_1|_\Phi; \Phi_2}\ \textsf{psem-atm-out}$$

$$\frac{\Phi \vdash^? A^- \rightharpoonup \rho; \Phi'}{\Phi \vdash^? A^- \times B^+ \rightharpoonup \rho; \Phi'}\ \textsf{psem-type}^+\textsf{-l}$$

$$\frac{\Phi \vdash A^- \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1 \vdash^? B^+ \rightharpoonup \rho_2; \Phi_2}{\Phi \vdash^? A^- \times B^+ \rightharpoonup \rho_2\rho_1; \Phi_2}\ \textsf{psem-type}^+\textsf{-r}$$

Figure 2.6: Partial semantics of LF

that computes a partial substitution $\Phi' \vdash \rho : \Phi$. The partial semantics is defined by adding the rules of Fig. 2.6. The important rules are psem-atm-in and psem-atm-out that stop execution before executing a matching between inputs and outputs respectively. The other rules simply propagate a partial substitution.

We extend the notion of mode-correct derivation to partial derivations and prove that well-moded programs have mode-correct derivations.

Definition 2.20 *A derivation $\Phi \vdash^? A \rightharpoonup \rho; \Phi'$ is mode-correct if the following conditions hold:*

1. *every subderivation $\Phi_0 \vdash A_0 \Longrightarrow \rho_0; \Phi_0'$ or $\Phi_0 \vdash^? A_0 \rightharpoonup \rho_0; \Phi_0'$ satisfies $\cdot \vdash \Phi_0$ decl and $\uparrow\Phi_0 \vdash A_0$ $\mathsf{goal} > \nu$ for some ν;*

2. *every subderivation $\Phi_0 \vdash S_2 =: S_1 \Longrightarrow \rho_0; \Phi_0'$ (resp. $\Phi_0 \vdash S_2 =: S_1 \Longrightarrow \rho_0; \Phi_0'$) satisfies $\uparrow\Phi_0 \vdash_{\mathsf{grd}} S_2$ and $\uparrow\Phi_0 \vdash S_1 > \nu$ for some ν;*

3. *every subderivation of the form*

$$\frac{\underline{\forall}c{:}\Pi\Gamma_0.A_0^+ \to a{\cdot}\check{S}_0\,\hat{S}_0 \in \Sigma, \Phi}{\Phi \vdash^? a{\cdot}\check{S}\,\hat{S} \rightharpoonup \cdot; \Phi}\ \textit{psem-atm-in}$$

   satisfies $\uparrow\Phi \vdash_{\mathsf{grd}} \check{S}$ and $\uparrow\Phi \vdash \check{S}? > \nu$ for some ν;

4. *every subderivation of the form*

$$\frac{\underline{\forall}c{:}\Pi\Gamma_0.A_0^+ \to a{\cdot}\check{S}_0\,\hat{S}_0 \in \Sigma, \Phi \qquad \Phi, \exists\Gamma_0 \vdash \check{S} =: \check{S}_0 \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1 \vdash \rho_1 A_0^+ \Longrightarrow \rho_2; \Phi_2}{\Phi \vdash^? a{\cdot}\check{S}\,\hat{S} \rightharpoonup \rho_2\rho_1|_\Phi; \Phi_2}\ \textit{psem-atm-out}$$

$\boxed{\forall^? \Gamma \vdash A}$

$$\frac{\vec{x}{:}A_0^+ \vdash \underline{\forall}\Gamma_1 \qquad \vec{x} \text{ fresh} \qquad \forall^? \Gamma, \forall \Gamma_0, \underline{\forall}\Gamma_1 \vdash a{\cdot}S}{\forall^? \Gamma \vdash \Pi\Gamma_0.A_0^+ \to a{\cdot}S}\ \textsf{osem-type}^-$$

$$\frac{\underline{\forall}c{:}\Pi\Gamma_0.A_0^+ \to a{\cdot}S_0 \in \Sigma, \Gamma \qquad \forall^? \Gamma \vdash \rho : \exists \Gamma_0 \qquad \rho S_0 = S \qquad \forall^? \Gamma \vdash \rho A_0^+}{\forall^? \Gamma \vdash a{\cdot}S}\ \textsf{osem-atm}$$

$$\frac{}{\forall^? \Gamma \vdash \mathbf{1}}\ \textsf{osem-one}$$

$$\frac{\forall^? \Gamma \vdash A^- \qquad \forall^? \Gamma \vdash B^+}{\forall^? \Gamma \vdash A^- \times B^+}\ \textsf{osem-type}^+$$

Figure 2.7: Oracle semantics for LF

satisfies $\uparrow\Phi_2 \vdash_{\mathsf{grd}} \rho\hat{S}$ *and* $\uparrow\Phi_2 \vdash \rho\hat{S}_0 > \nu$ *for some* ν.

Lemma 2.21 *Assume that* $\vdash \Sigma$ decl, $\cdot \vdash \uparrow\Phi$ decl *and* $\uparrow\Phi \vdash A$ goal $> \nu$ *for some* ν *(where* A *is a positive, negative, or atomic type). Then any derivation* $\Phi \vdash^? A \rightharpoonup \rho; \Phi'$ *is mode correct.*

Proof: By induction on the semantics derivation and case analysis on the last rule used. The proof proceeds similarly to Lemma 2.18. □

2.3.2 Completeness of the moded semantics

Finally, we show that, for well-moded programs, the moded semantics is complete with respect to an *oracle semantics*. The oracle semantics is given by the judgment

$$\forall^? \Gamma \vdash A$$

meaning basically that A is inhabited in Γ. The rules are given in Fig. 2.7. The oracle gives the value of ρ in rule osem-atm.

The completeness of the moded semantics with respect to the oracle semantics is expressed by the following theorem. We need a lemma on the completeness of the matching algorithm, which follows from Assumption 2.4.

Lemma 2.22 *Let* S_1 *and* S_2 *be well-typed spines under context* $|\Phi|$ *such that* S_1 *is in the pattern fragment and* S_2 *is ground. Assume* $\forall^? \Gamma \vdash \rho : \Phi$ *such that* $\rho S_1 = S_2$. *Then there exists* ρ_0 *and* ρ_1 *such that* $\Phi \vdash S_2 =: S_1 \Longrightarrow \rho_0; \Phi_1$, $\forall^? \Gamma ? \rho_1 : \Phi_1$ *and* $\rho = \rho_1\rho_0$.

Theorem 2.23 *Assume that* $\vdash \Sigma$ decl, $\cdot \vdash \Phi$ decl, $\uparrow\Phi \vdash A$ goal $> \nu$, *and* $\forall^? \Gamma \vdash \rho : \Phi$. *If* $\forall^? \Gamma \vdash \rho A$, *then there exists* ρ_0 *and* ρ_1 *such that* $\Phi \vdash A \Longrightarrow \rho_0; \Phi'$, $\forall^? \Gamma \vdash \rho_1 : \Phi'$ *and* $\rho = \rho_1\rho_0$.

Proof: By induction on the derivation in the oracle semantics and case analysis on the last rule applied.

osem-type$^-$ We have the derivation

$$\frac{\vec{x}{:}A_0^+ \vdash \underline{\forall}\Gamma_1 \qquad \vec{x} \text{ fresh} \qquad \forall^?\Gamma, \forall\rho\Gamma_0, \underline{\forall}\rho\Gamma_1 \vdash a{\cdot}\rho S}{\forall^?\Gamma \vdash \rho(\Pi\Gamma_0.A_0^+ \to a{\cdot}S)}$$

By inversion on the derivation $\uparrow\Phi \vdash \Pi\Gamma_0.A_0^+ \to a{\cdot}S$ $\mathsf{goal} > \nu$ we have $\uparrow\Phi, \forall\Gamma_0 \vdash a{\cdot}S$ $\mathsf{goal} > \nu$ and $\uparrow\Phi, \forall\Gamma_0 \vdash A_0^+$ decl. From the latter, $\cdot \vdash \Phi, \forall\Gamma_0, \underline{\forall}\Gamma_1$ decl. Let $\rho' = \rho, \mathrm{id}_{\forall\Gamma_0, \underline{\forall}\Gamma_1}$. Then $\forall^?\Gamma, \forall\rho\Gamma_0, \underline{\forall}\rho\Gamma_1 \vdash \rho' : \Phi, \forall\Gamma_0, \underline{\forall}\Gamma_1$.

By IH, there exists ρ_0 and ρ_1 such that $\uparrow\Phi, \forall\Gamma_0, \underline{\forall}\Gamma_1 \vdash a{\cdot}S \Longrightarrow \rho_0; \Phi'$, $\forall^?\Gamma, \forall\rho_0\Gamma_0, \underline{\forall}\rho_0\Gamma_1 \vdash \rho_1 : \Phi'$, and $\rho' = \rho_1\rho_0$. The context Φ' must be of the form $\Phi'', \forall\rho_0\Gamma_0, \underline{\forall}\rho_0\Gamma_1$. Then $\rho_0|_\Phi$ and $\rho_1|_{\Phi'}$ are the desired substitutions.

osem-atm We have the derivation

$$\frac{\begin{array}{c}\underline{\forall}c{:}\Pi\Gamma_0.A_0^+ \to a{\cdot}S_0 \in \Sigma, \Gamma \\ \forall^?\Gamma \vdash \rho_0 : \exists\Gamma_0 \qquad \rho_0 S_0 = \rho S \qquad \forall^?\Gamma \vdash \rho_0 A_0^+\end{array}}{\forall^?\Gamma \vdash \rho(a{\cdot}S)}$$

We have to show that we can find a derivation in the operational semantics for type $a{\cdot}S$ in context Φ.

We first show that there is a clause in Σ, Φ corresponding to h. Let $A^- = \Pi\Gamma_0.A_0^+ \to a{\cdot}S_0$. Constant h is declared either in Σ or Γ. In the latter case, h is also declared in Φ with a type A'^- such that $\rho A'^- = A^-$. In the former case, we can also assume that h has type A'^- satisfying $\rho A'^- = A^-$, by setting $A'^- = A^-$ since ρ behaves like the identity on A^-. In both cases, A'^- has the form $\Pi\Gamma'_0.{A_0'}^+ \to a{\cdot}S'_0$, and $\rho A'^- = A^-$.

From $\forall^?\Gamma \vdash \rho : \Phi$ and $\forall^?\Gamma \vdash \rho_0 : \exists\Gamma?$ we have $\forall^?\Gamma \vdash \rho, \rho_0 : \Phi, \exists\Gamma_0$. Let $\rho' = \rho, \rho_0$. Let $S = \check{S}\,\hat{S}$, $S_0 = \check{S}_0\,\hat{S}_0$, and $S'_0 = \check{S}'_0\,\hat{S}'_0$, according to the mode declaration of a. By inversion on the mode-checking derivation of A and h we have that $\check{S}'$ is ground and $\check{S}'_0$ is in the pattern fragment. We have $\rho_0 S_0 = \rho_0\rho S'_0 = (\rho, \rho_0)S'_0 = \rho' S'_0$ and $\rho S = \rho' S$. By Lemma 2.22, there exist ρ_1 and ρ'_1 such that $\Phi, \exists\Gamma_0 \vdash \check{S} =: \check{S}'_0 \Longrightarrow \rho_1; \Phi_1$, $\forall^?\Gamma_0\rho'_1 : \Phi_1$ and $\rho' = \rho'_1\rho_1$.

Note that $\rho_0 A_0^+ = \rho_0\rho {A_0'}^+ = (\rho, \rho_0){A_0'}^+ = \rho' {A_0'}^+ = \rho'_1\rho_1 {A_0'}^+$. By IH, there exists ρ_2 and ρ'_2 such that $\Phi_1 \vdash \rho_1 {A_0'}^+ \Longrightarrow \rho_2; \Phi_2$, $\forall^?\Gamma \vdash \rho'_2 : \Phi_2$ and $\rho'_1 = \rho'_2\rho_2$.

We have $\rho' = \rho'_2\rho_2\rho_1$. Then, We have $\rho'_2\rho_2\rho_1\hat{S}'_0 = \rho'_2\rho_2\rho_1\hat{S}$. By Lemma 2.22, there exist ρ_3 and ρ'_3 such that $\Phi_2 \vdash \rho_2\rho_1\hat{S}'_0 =: \rho_2\rho_1\hat{S} \Longrightarrow \rho_3; \Phi_3$, $\forall^?\Gamma \vdash \rho'_3 : \Phi_3$ and $\rho'_2 = \rho'_3\rho_3$. Hence, $\Phi_3 \vdash \rho_3\rho_2\rho_1 : \Phi, \exists\Gamma_0$ and $\forall^?\Gamma \vdash \rho'_3 : \Phi_3$. Then, $\rho_3\rho_2\rho_1|_\Phi$ and ρ'_3 are the desired substitutions.

osem-one Trivial.

osem-type$^+$ We have the derivation

$$\frac{\forall^?\Gamma \vdash \rho A^- \qquad \forall^?\Gamma \vdash \rho B^+}{\forall^?\Gamma \vdash \rho(A^- \times B^+)}$$

By IH on the derivation of ρA^- there exists ρ_0 and ρ_1 such that $\Phi \vdash A \Longrightarrow \rho_0; \Phi_1$, $\forall^?\Gamma \vdash \rho_1 : \Phi_1$ and $\rho = \rho_1\rho_0$. By Theorem 2.16, $\cdot \vdash \Phi_1$ decl. By IH on the derivation of ρB^+ there exists ρ'_0 and ρ'_1 such that $\Phi_1 \vdash B^+ \Longrightarrow \rho'_0; \Phi_2$, $\forall^?\Gamma \vdash \rho'_1 : \Phi_2$ and $\rho_1 = \rho'_1\rho'_0$. Then, $\rho'_0\rho_0$ and ρ'_1 are the desired substitutions.

□

3 Mode checking in CLF

In this section we extend the development from the previous section to a significant fragment of CLF that combines forward chaining and backward chaining. We call the fragment of CLF studied in this report Mini-CLF. With respect to full CLF, Mini-CLF does not feature affine hypotheses nor additive conjunctions. Adding these features does not significantly change the definition of the mode checker nor the proof of correctness.

We follow the same template as in Sect. 2. In Sect. 3.1 we present the syntax and operational semantics of Mini-CLF. In Sect. 3.2 we define the mode checking algorithm. Finally, in Sect. 3.3 we prove the correctness of the mode checker.

3.1 Syntax and semantics of Mini-CLF

Mini-CLF is a restriction of CLF where additive conjunctions and affine types are removed. Alternatively, Mini-CLF can be seen as an extension of LF with linear and monadic types.

For completeness, we first present the syntax of full CLF:

$$
\begin{array}{rll}
K & ::= \mathsf{type} \mid \Pi x{:}A^-.K & \text{(Kinds)} \\
A^- & ::= a\cdot S' \mid A_1^- \mathbin{\&} A_2^- \mid \Pi p{:}A^+.A^- \mid \{A^+\} & \text{(Negative types)} \\
A^+ & ::= \mathbf{1} \mid \exists p{:}A^+.A^+ \mid\ !A^- \mid @A^- \mid \downarrow A^- & \text{(Positive types)} \\
N & ::= H\cdot S \mid \langle N_1, N_2\rangle \mid \widehat{\lambda} p.N \mid \{E\} & \text{(Negative terms)} \\
H & ::= x \mid c & \text{(Heads)} \\
S' & ::= () \mid N; S & \text{(Type spines)} \\
S & ::= () \mid \pi_1; S \mid \pi_2; S \mid M; S & \text{(Spines)} \\
E & ::= M \mid \mathsf{let}\ \epsilon\ \mathsf{in}\ E & \text{(Expressions)} \\
\epsilon & ::= \cdot \mid \delta \mid \epsilon_1; \epsilon_2 & \text{(Traces)} \\
\delta & ::= \{p\} = H\cdot S & \text{(Steps)} \\
p & ::= \mathbf{1} \mid \langle p_1, p_2\rangle \mid\ !x \mid @x \mid \downarrow x & \text{(Patterns)} \\
M & ::= \mathbf{1} \mid \langle M_1, M_2\rangle \mid\ !N \mid @N \mid \downarrow N & \text{(Positive terms)} \\
\Sigma & ::= \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A^- & \text{(Signatures)}
\end{array}
$$

The constructions in gray are not included in Mini-CLF.

Expressions are defined using traces for denoting the terms corresponding to the monadic fragment, following [12, 17].

CLF has three different injections from negative into positive types (the last three productions of A^+) and three corresponding flags for variable and residual terms. They have the following meaning:

- $!A^-$ indicates that A^- is a *persistent* type. Pattern variables of this type have the form $!x$ and residual terms have the form $!N$.

  In the mode checking rules, we distinguish between dependent and non-dependent occurrences of persistent types. Dependent goals are solved in the semantics by unification (or matching), while non-dependent goals are solved by search.

$\boxed{\Gamma \vdash A^- : \mathsf{type}}$

$$\frac{\Gamma \vdash \Gamma_1 \qquad \Gamma, \Gamma_1 \vdash A_1^+ : \mathsf{type} \qquad \Gamma, \Gamma_1 \vdash \Gamma_2 \qquad \Gamma, \Gamma_1, \Gamma_2 \vdash A_2^+ : \mathsf{type}}{\Gamma \vdash \Pi\Gamma_1.A_1^+ \multimap \{\exists \Gamma_2.A_2^+\} : \mathsf{type}}$$

$\boxed{\Gamma \vdash A^+ : \mathsf{type}}$

$$\frac{}{\Gamma \vdash \mathbf{1} : \mathsf{type}} \qquad \frac{\Gamma \vdash A^- : \mathsf{type} \qquad \Gamma \vdash B^+ : \mathsf{type}}{\Gamma \vdash\, !A^- \otimes B^+ : \mathsf{type}} \qquad \frac{\Gamma \vdash A^- : \mathsf{type} \qquad \Gamma \vdash B^+ : \mathsf{type}}{\Gamma \vdash \downarrow A^- \otimes B^+ : \mathsf{type}}$$

Figure 3.1: Additional typing rules for Mini-CLF

- $@A^-$ indicates that A^- is a (non-dependent) *affine* type. Pattern variables of this type have the form $@x$ and residual terms have the form $@N$. Affine types are never dependent.
- $\downarrow A^-$ indicates that A^- is a (non-dependent) *linear* type. Pattern variables of this type have the form $\downarrow x$ and residual terms have the form $\downarrow N$. Linear types are never dependent.

Mini-CLF Following the development of the previous section, we give a presentation of Mini-CLF suitable for defining the operational semantics and mode checker. This alternative presentation makes explicit the distinction between dependent and non-dependent arguments in products. Also, the term language is left unspecified, since the semantics does not construct a proof term.

The syntax of Mini-CLF is as follows:

$$\begin{aligned}
K &::= \Pi\Gamma.\mathsf{type} && \text{(Kinds)}\\
P &::= a{\cdot}S \mid \{\exists\Gamma.A^+\} && \text{(Type heads)}\\
A^- &::= \Pi\Gamma.A_0^+P && \text{(Negative types)}\\
A^+ &::= \mathbf{1} \mid \downarrow A^- \otimes A^+ \mid\, !A^- \otimes A^+ && \text{(Positive types)}\\
S &::= () \mid N; S && \text{(Spines)}\\
\Gamma, \Delta &::= \cdot \mid \Gamma, x{:}A^- && \text{(Contexts)}\\
\Sigma &::= \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A^- && \text{(Signatures)}
\end{aligned}$$

A positive type is essentially a sequence of linear and persistent hypotheses. All dependent hypotheses are persistent; no linear dependencies are allowed. Types whose head is of the form $a{\cdot}S$ are called atomic types, while types whose head is of the form $\{\exists\Gamma.A^+\}$ are called monadic types.

Typing rules Since dependencies are only persistent, the typing rules for types only depend on the persistent context, while typing rules for terms depend on a persistent and linear context. The typing judgments are the same as for LF. The typing rules include all the rules from Fig. 2.1 (except for the rules for positive types) and rules of Fig. 3.1.

Operational semantics It is defined using mixed-prefix contexts and purely linear contexts (i.e., contexts that contain only linear hypotheses). The operational semantics is defined by the following

judgments:

$$\Phi; \Delta \vdash A^- \Longrightarrow \rho; \Phi'$$
$$\Phi; \Delta \vdash a{\cdot}S \Longrightarrow \rho; \Phi'$$
$$\Phi; \Delta \vdash \{\exists \Phi. A^+\} \Longrightarrow \rho; \Phi'$$
$$\Phi; \Delta \vdash A^+ \Longrightarrow \rho; \Phi'$$
$$\Phi; \Delta \rightsquigarrow \rho; \Phi'; \forall^? \Gamma; \Delta'$$

Persistent hypotheses are placed in a mixed-prefix context to differentiate between parameters and existential hypotheses. However, since there are no linear dependencies, the linear hypotheses are placed in a plain context.

The operational semantics is defined by the rules of Fig. 3.2. In rule sem-type$^-$ we use the judgment $\vec{x}{:}A^- \Longrightarrow \underline{\forall}\Gamma, \Delta$ to construct a context from a sequence of variables and a positive type; it is defined by the following rules:

$$\frac{}{\cdot{:}\mathbf{1} \Longrightarrow \cdot} \qquad \frac{\vec{x}{:}B^+ \Longrightarrow \underline{\forall}\Gamma, \Delta}{x\,\vec{x}{:}{\downarrow}A^- \otimes B^+ \Longrightarrow \underline{\forall}\Gamma; (x{:}A^-, \Delta)} \qquad \frac{\vec{x}{:}B^+ \Longrightarrow \underline{\forall}\Gamma, \Delta}{x\,\vec{x}{:}!A^- \otimes B^+ \Longrightarrow (\underline{\forall}x{:}A^-, \underline{\forall}\Gamma); \Delta}$$

In rule sem-atm, we write $\Delta \setminus\!\setminus h$ to mean the context obtained by removing the declaration of h (if h is declared in Δ; otherwise this is the same as Δ).

Forward chaining is defined by rules sem-stop, sem-step, and sem-comp. They correspond to the trace constructors in CLF: empty trace, a single step, and trace composition. We do not specify any procedure for choosing which hypothesis to apply in rule sem-step nor any stopping criterion in rule sem-stop. The judgment $\Phi; \Delta \rightsquigarrow \rho; \Phi'; \forall^? \Gamma; \Delta'$ means that $\Phi' \vdash \rho : \Phi$, and the context pair $\Phi; \Delta$ is transformed into $\Phi', \forall^? \Gamma; \Delta'$ by the forward chaining steps. The context $\forall^? \Gamma$ contains the new persistent binders introduced during forward chaining, while Δ' contains new linear hypotheses (as well as unused hypotheses from Δ).

The pattern fragment is extended to expressions: an expression $\mathsf{let}\ \delta_1; \ldots; \delta_n\ \mathsf{in}\ M$ is in the pattern fragment if the following conditions hold [12]:

1. at most one step whose head is an existentially quantified variable and is applied to a pattern;
2. every other step is in the pattern fragment for LF; and
3. M is a sequence of variables.

These conditions ensures that the matching algorithm is sound and complete.

3.2 Mode checking

In this section, we define the mode checker for Mini-CLF by extending the mode checker defined in Sect. 2.2.

We use the same judgments as in Sect. 2.2 for mode checking with the addition of judgments to check forward chaining declarations:

- $\nu \vdash A^-$ fwd checks that A^- is a valid declaration to be used in forward chaining; the head type A^- must be monadic.

$\boxed{\Phi; \Delta \vdash A^- \Longrightarrow \rho; \Phi'}$

$$\frac{\begin{array}{c}\vec{x}{:}A^+ \Longrightarrow \underline{\forall}\Gamma_0; \Delta_0 \qquad \vec{x} \text{ fresh} \\ \Phi, \forall\Gamma, \underline{\forall}\Gamma_0; \Delta, \Delta_0 \vdash P \Longrightarrow \rho; \Phi_1, \forall^?\Gamma_1 \qquad \#\Gamma_1 = \#(\Gamma, \Gamma_0)\end{array}}{\Phi; \Delta \vdash \Pi\Gamma.A^+ \multimap P \Longrightarrow \rho|_\Phi; \Phi_1} \text{ sem-type}^-$$

$\boxed{\Phi; \Delta \vdash P \Longrightarrow \rho; \Phi'}$

$$\frac{\begin{array}{c}h : \Pi\Gamma_0.A_0^+ \multimap a{\cdot}\check{S}_0\,\hat{S}_0 \in \Sigma, \Phi, \Delta \qquad \Phi, \exists\Gamma_0 \vdash \check{S} =: \check{S}_0 \Longrightarrow \rho_1; \Phi_1 \\ \Phi_1; (\Delta \setminus\!\setminus h) \vdash \rho_1 A_0^+ \Longrightarrow \rho_2; \Phi_2 \qquad \Phi_2 \vdash \rho_2\rho_1\hat{S}_0 =: \rho_2\rho_1\hat{S} \Longrightarrow \rho_3; \Phi_3\end{array}}{\Phi; \Delta \vdash a{\cdot}\check{S}\,\hat{S} \Longrightarrow \rho_3\rho_2\rho_1|_\Phi; \Phi_3} \text{ sem-atm}$$

$$\frac{\begin{array}{c}\Phi; \Delta \rightsquigarrow \rho_1; \Phi_1; \forall^?\Gamma_1; \Delta_1 \\ \Phi_1, \forall^?\Gamma_1, \exists\rho_1\Gamma_0; \Delta_1 \vdash \rho_1 A_0^+ \Longrightarrow \rho_2; \Phi_2, \forall^?\Gamma_2 \qquad \#\Gamma_2 = \#\Gamma_1\end{array}}{\Phi; \Delta \vdash \{\exists\Gamma_0.A_0^+\} \Longrightarrow \rho_2|_{\Phi_1}\rho_1; \Phi_2} \text{ sem-mon}$$

$\boxed{\Phi; \Delta \vdash A^+ \Longrightarrow \rho; \Phi'}$

$$\frac{}{\Phi; \cdot \vdash \mathbf{1} \Longrightarrow \mathrm{id}_\Phi; \Phi} \text{ sem-one}$$

$$\frac{\Phi; \Delta_1 \vdash A^- \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1; \rho_1\Delta_2 \vdash \rho_1 B^+ \Longrightarrow \rho_2; \Phi_2}{\Phi; \Delta_1, \Delta_2 \vdash \downarrow A^- \otimes B^+ \Longrightarrow \rho_2\rho_1; \Phi_2} \text{ sem-prod-lin}$$

$$\frac{\Phi; \cdot \vdash A^- \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1; \rho_1\Delta \vdash \rho_1 B^+ \Longrightarrow \rho_2; \Phi_2}{\Phi; \Delta \vdash !A^- \otimes B^+ \Longrightarrow \rho_2\rho_1; \Phi_2} \text{ sem-prod-bang}$$

$\boxed{\Phi; \Delta \rightsquigarrow \rho; \Phi'; \forall^?\Gamma; \Delta'}$

$$\frac{}{\Phi; \Delta \rightsquigarrow \mathrm{id}_\Phi; \Phi; \cdot; \Delta} \text{ sem-stop}$$

$$\frac{\begin{array}{c}h : \Pi\Gamma.A^+ \multimap \{\exists\Gamma_0.A_0^+\} \in \Sigma, \Phi, \Delta \qquad \Phi, \exists\Gamma; \Delta \setminus\!\setminus h \vdash A^+ \Longrightarrow \rho; \Phi_1 \\ \vec{x}{:}\rho A_0^+ \Longrightarrow \underline{\forall}\Gamma_1; \Delta_1 \qquad \vec{x} \text{ fresh}\end{array}}{\Phi; \Delta, \Delta_0 \rightsquigarrow \rho|_\Phi; \Phi_1; \forall\rho\Gamma_0, \underline{\forall}\Gamma_1; \Delta_0, \Delta_1} \text{ sem-step}$$

$$\frac{\begin{array}{c}\Phi_0; \Delta_0 \rightsquigarrow \rho_1; \Phi_1; \forall^?\Gamma_1; \Delta_1 \\ \Phi_1, \forall^?\Gamma_1; \Delta_1 \rightsquigarrow \rho_2; \Phi_2, \forall^?\Gamma_1'; \forall^?\Gamma_2; \Delta_2 \qquad \#\Gamma_1 = \#\Gamma_1'\end{array}}{\Phi_0; \Delta_0 \rightsquigarrow \rho_2|_{\Phi_1}\rho_1; \Phi_2; \forall^?\Gamma_1', \forall^?\Gamma_2'; \Delta_2} \text{ sem-comp}$$

Figure 3.2: Operational semantics of Mini-CLF

$\boxed{\nu \vdash A^- \ \mathsf{fwd}}$

$$\frac{\nu, \exists^{\mathsf{u}}\Gamma_1 \vdash A_1^+ \ \mathsf{goal} > \nu', \exists^{\mathsf{g}}\Gamma_1 \qquad \nu', \exists^{\mathsf{g}}\Gamma_1, \forall\Gamma_2 \vdash A_2^+ \ \mathsf{fwd}}{\nu \vdash \Pi\Gamma_1.A_1^+ \multimap \{\exists\Gamma_2.A_2^+\} \ \mathsf{fwd}} \ \mathsf{mc\text{-}fwd}$$

$\boxed{\nu \vdash A^+ \ \mathsf{fwd}}$

$$\frac{}{\nu \vdash \mathbf{1} \ \mathsf{fwd}} \ \mathsf{mcf\text{-}one}$$

$$\frac{\nu \vdash A^- \ \mathsf{decl} \qquad \nu \vdash B^+ \ \mathsf{fwd}}{\nu \vdash \downarrow A^- \otimes B^+ \ \mathsf{fwd}} \ \mathsf{mcf\text{-}lin} \qquad \frac{\nu \vdash A^- \ \mathsf{decl} \qquad \nu \vdash B^+ \ \mathsf{fwd}}{\nu \vdash !A^- \otimes B^+ \ \mathsf{fwd}} \ \mathsf{mcf\text{-}pers}$$

$\boxed{\nu \vdash A \ \mathsf{goal} > \nu'}$

$$\frac{\nu, \exists^{\mathsf{u}}\Gamma \vdash A^+ \ \mathsf{goal} > \nu', \exists^{\mathsf{g}}\Gamma}{\nu \vdash \{\exists\Gamma.A^+\} \ \mathsf{goal} > \nu'} \ \mathsf{mcg\text{-}mon}$$

$$\frac{\nu \vdash B^+ \ \mathsf{goal} > \nu' \qquad \nu' \vdash A^- \ \mathsf{goal} > \nu''}{\nu \vdash !A^- \otimes B^+ \ \mathsf{goal} > \nu''} \ \mathsf{mcg\text{-}pers}$$

$$\frac{\nu \vdash B^+ \ \mathsf{goal} > \nu' \qquad \nu' \vdash A^- \ \mathsf{goal} > \nu''}{\nu \vdash \downarrow A^- \otimes B^+ \ \mathsf{goal} > \nu''} \ \mathsf{mcg\text{-}lin}$$

$\boxed{\nu \vdash A^- \ \mathsf{decl}}$

$$\frac{\nu \vdash \Pi\Gamma_1.A_1^+ \multimap \{\exists\Gamma_2.A_2^+\} \ \mathsf{fwd}}{\nu \vdash \Pi\Gamma_1.A_1^+ \multimap \{\exists\Gamma_2.A_2^+\} \ \mathsf{decl}}$$

Figure 3.3: Additional mode checking rules for Mini-CLF

- $\nu \vdash A^+ \ \mathsf{fwd}$ checks that all negative types in A^+ are valid declarations. When applying a rule of the form $\Pi\Gamma_1.A_1^+ \multimap \{\exists\Gamma_2.A_2^+\}$, the context is extended with Γ_2 and the components (negative types) in A_2^+. The judgment $\forall\Gamma_2 \vdash A_2^+ \ \mathsf{fwd}$ checks that the components of A_2^+ are well moded.

The mode checking rules are those of Fig. 2.5 plus the rules of Fig. 3.3.

Rule mc-fwd checks that monadic types are valid to be used in forward chaining. Similar to rule mc-bwd, we check that goals in A_1^+ force the dependent arguments to be ground. Then, we check that A_2^+ consists of valid declarations. The rules mcf-one, mcf-lin, mcf-pers simply iterate over a positive type, checking that all components are valid declarations.

The rule mcg-mon checks that a monadic head type is a valid goal, by checking that solving the non-dependent goals in A^+ grounds the dependent variables in Γ. Rules mcg-pers and mcg-lin simply iterate over a positive type.

Finally, rule mcd-fwd extends the judgment for mode checking declarations to monadic types.

3.3 Correctness of the mode checker in CLF

We show that the mode checker for Mini-CLF is correct in the same way as in Sect. 2.3.

The mode checker satisfies all the properties of Sect. 2.3 and the following property extending Lemma 2.13 to forward chaining.

Lemma 3.1 (Substitution lemma for mode checking forward declarations) *If* $\nu \vdash A$ fwd *and* $\nu' \vdash \rho : \nu$*, then* $\nu' \vdash \rho A$ fwd*.*

The following theorem states that the mode checker is a form of abstract interpretation over the semantics.

Theorem 3.2 *Assume that* $\vdash \Sigma$ decl, $\cdot \vdash \Phi$ decl, *and* $\uparrow\Phi \vdash \underline{\forall}\Delta$ decl.

1. *If* $\uparrow\Phi \vdash A$ goal $> \nu$*, where* A *is either a positive, negative, or atomic type, and* $\Phi; \Delta \vdash A \Longrightarrow \rho; \Phi'$*, then* $\Phi' \vdash \rho : \nu$*.*
2. *If* $\Phi; \Delta \rightsquigarrow \rho; \Phi'; \forall^?\Gamma; \Delta'$ *then* $\uparrow\Phi' \vdash \forall^?\Gamma, \underline{\forall}\Delta'$ decl*.*
3. *If* $\Phi \vdash S_2 =: S_1 \Longrightarrow \rho; \Phi'$*,* $\uparrow\Phi \vdash_{\mathsf{grd}} S_2$*, and* $\uparrow\Phi \vdash S_1 > \nu$*, then* $\Phi' \vdash \rho : \nu$*.*

Proof: We proceed by induction on the semantic derivation and case analysis on the last rule used.

sem-type$^-$ We have the derivation

$$\frac{\vec{x}{:}A^+ \Longrightarrow \underline{\forall}\Gamma_0; \Delta_0 \qquad \vec{x} \text{ fresh} \qquad \Phi, \forall\Gamma, \underline{\forall}\Gamma_0; \Delta, \Delta_0 \vdash P \Longrightarrow \rho; \Phi_1, \forall^?\Gamma_1 \qquad \#\Gamma_1 = \#(\Gamma, \Gamma_0)}{\Phi; \Delta \vdash \Pi\Gamma.A^+ \multimap P \Longrightarrow \rho|_\Phi; \Phi_1}$$

By inversion on $\uparrow\Phi \vdash \Pi\Gamma.A^+ \multimap P$ goal $> \nu$, there exists ν' such that $\uparrow\Phi \vdash P$ goal $> \nu$, and $\uparrow\Phi, \underline{\forall}\Gamma \vdash A^+$ decl. From the latter and $\vec{x}{:}\rho_0 A^+ \Longrightarrow \underline{\forall}\Gamma_0, \Delta_0$, we have $\cdot \vdash \Phi, \forall\Gamma, \underline{\forall}\Gamma_0$ decl and $\uparrow\Phi, \forall\Gamma, \forall\Gamma_0 \vdash \Delta, \Delta_0$ decl. By IH, $\Phi_1, \forall^?\Gamma_1 \vdash \rho : \nu, \mathrm{id}_{\forall\Gamma, \underline{\forall}\Gamma_0}$. The result follows by restricting the domain of ρ to ν.

sem-atm Similar to the corresponding rule in Theorem 2.16.

sem-mon We have the derivation

$$\frac{\Phi; \Delta \rightsquigarrow \rho_1; \Phi_1; \forall^?\Gamma_1; \Delta_1 \qquad \Phi_1, \forall^?\Gamma_1, \exists\rho_1\Gamma_0; \Delta_1 \vdash \rho_1 A_0^+ \Longrightarrow \rho_2; \Phi_2, \forall^?\Gamma_2 \qquad \#\Gamma_2 = \#\Gamma_1}{\Phi; \Delta \vdash \{\exists\Gamma_0.A_0^+\} \Longrightarrow \rho_2|_{\Phi_1}\rho_1; \Phi_2}$$

By IH on the first premise, $\uparrow\Phi_1 \vdash \forall^?\Gamma_1, \underline{\forall}\Delta_1$ decl. By inversion on $\uparrow\Phi \vdash \{\exists\Gamma_0.A_0^+\}$ goal $> \nu$ we have $\uparrow\Phi, \exists^{\mathsf{u}}\Gamma_0 \vdash A_0^+$ goal $> \nu, \exists^{\mathsf{g}}\Gamma_0$. By weakening (Lemma 2.11), $\uparrow\Phi, \forall\Gamma_1, \exists^{\mathsf{u}}\Gamma_0 \vdash A_0^+$ goal $> \nu, \forall\Gamma_1, \exists^{\mathsf{g}}\Gamma_0$ Extending ρ_1 we have $\Phi_1, \forall^?\Gamma_1, \exists\Gamma_0 \vdash \rho_1, \mathrm{id}_{\forall^?\Gamma_1, \exists\Gamma_0} : \Phi, \forall^?\Gamma_1, \exists\Gamma_0$. Let $\rho_1' = \rho_1, \mathrm{id}_{\forall^?\Gamma_1, \exists\Gamma_0}$. By Lemma 2.13, there exists ν' such that $\uparrow\Phi_1, \forall^?\Gamma, \exists^{\mathsf{u}}\Gamma_0 \vdash \rho_1' A_0^+$ goal $> \nu'$ and $\nu' \vdash \rho_1' : \nu, \forall\Gamma_1, \exists^{\mathsf{g}}\Gamma_0$. Then ν' is of the form $\nu'', \forall\Gamma_1, \exists^{\mathsf{g}}\Gamma_0$ for some ν'' and $\nu'' \vdash \rho_1 : \nu$. Since $\rho_1' A_0^+ = \rho_1 A_0^+$, we can apply IH to obtain $\Phi_2, \forall^?\Gamma_2 \vdash \rho_2 : \nu'$. Then $\Phi_2 \vdash \rho_2|_{\Phi_1} : \nu''$ and $\Phi_2 \vdash \rho_2|_{\Phi_1}\rho_1 : \nu$ as desired.

sem-one Follows immediately.

sem-prod-lin We have the following derivation

$$\frac{\Phi; \Delta_1 \vdash A^- \Longrightarrow \rho_1; \Phi_1 \qquad \Phi_1; \rho_1\Delta_2 \vdash \rho_1 B^+ \Longrightarrow \rho_2; \Phi_2}{\Phi; \Delta_1, \Delta_2 \vdash \downarrow A^- \otimes B^+ \Longrightarrow \rho_2\rho_1; \Phi_2}$$

From $\uparrow\Phi \vdash \Delta_1, \Delta_2$ decl we have $\uparrow\Phi \vdash \Delta_1$ decl and $\uparrow\Phi \vdash \Delta_2$ decl. By inversion on $\uparrow\Phi \vdash \downarrow A^- \otimes B^+$ $\mathsf{goal} > \nu$ there exists ν' such that $\uparrow\Phi \vdash A^-$ $\mathsf{goal} > \nu'$ and $\nu' \vdash B^+$ $\mathsf{goal} > \nu$. By IH on A^-, $\Phi_1 \vdash \rho_1 : \nu'$. By Lemma 2.14, $\uparrow\Phi_1 \vdash \rho_1\Delta_2$ decl. By Lemma 2.13, there exists ν'' such that $\uparrow\Phi_1 \vdash \rho_1 B^+$ $\mathsf{goal} > \nu''$ and $\nu'' \vdash \rho_1 : \nu$. By IH on B^+, $\Phi_2 \vdash \rho_2 : \nu''$. Combining ρ_2 and ρ_1 we get $\Phi_2 \vdash \rho_2\rho_1 : \nu$ as desired.

sem-prod-bang Similar to the previous case.

sem-stop Follows immediately.

sem-step We have the following derivation

$$\frac{h : \Pi\Gamma.A^+ \multimap \{\exists\Gamma_0.A_0^+\} \in \Sigma, \Phi, \Delta \qquad \Phi, \exists\Gamma; \Delta \setminus\!\setminus h \vdash A^+ \Longrightarrow \rho; \Phi_1 \qquad \vec{x}{:}\rho A_0^+ \Longrightarrow \underline{\forall}\Gamma_1; \Delta_1 \qquad \vec{x} \text{ fresh}}{\Phi; \Delta, \Delta_0 \rightsquigarrow \rho|_\Phi; \Phi_1; \forall\rho\Gamma_0, \underline{\forall}\Gamma_1; \Delta_0, \Delta_1}$$

By inversion on the mode-checking derivation of the type of h and weakening (Lemma 2.11), we have $\uparrow\Phi \vdash \Pi\Gamma.A^+ \multimap \{\exists\Gamma_0.A_0^+\}$ decl. Again by inversion on the mode-checking derivation there exists ν' such that $\uparrow\Phi, \exists^{\mathsf{u}}\Gamma \vdash A^+$ $\mathsf{goal} > \nu', \exists^{\mathsf{g}}\Gamma$ and $\nu', \exists^{\mathsf{g}}\Gamma, \forall\Gamma_0 \vdash A_0^+$ fwd.

By IH on A^+, we have $\Phi_1 \vdash \rho : \nu', \exists^{\mathsf{g}}\Gamma$. Then, $\uparrow\Phi_1 \vdash \rho : \nu', \exists^{\mathsf{g}}\Gamma$ and, by substitution on the mode-checking derivation, we have $\uparrow\Phi_1 \vdash \rho A_0^+$ fwd. Note that this last judgment means checking that every component of ρA_0^+ is a valid declaration. It is then easy to see check that $\uparrow\Phi_1 \vdash \forall\rho\Gamma_0, \underline{\forall}\Gamma_1, \Delta_0, \Delta_1$ decl.

sem-comp Follows directly from two applications of the IH.

□

We can adapt directly the definition of mode-correct derivation (Def. 2.17) to Mini-CLF. The following theorem states that the well-moded programs have mode-correct derivations (similar to Lemma 2.18).

Theorem 3.3 *Assume that* $\vdash \Sigma$ decl, $\cdot \vdash \uparrow\Phi$ decl, $\uparrow\Phi \vdash \underline{\forall}\Delta$ decl, *and* $\uparrow\Phi \vdash A$ $\mathsf{goal} > \nu$ *for some* ν *(where* A *is a positive, negative, or atomic type). Then any derivation* $\Phi; \Delta \vdash A \Longrightarrow \rho; \Phi'; \Delta'$ *is mode correct.*

Proof: By induction on the semantics derivation and case analysis on the last rule used. All cases proceed similarly to Theorem 3.2. □

3.3.1 Partial semantics

We can define a partial semantics for Mini-CLF as we did in Sect. 2 and prove that derivations in the partial semantics are also mode correct.

We omit the development here as it is very similar to that of Sect. 2.3.1. The main difference is the addition of the linear context. In particular, the rules related to forward chaining do not change in the partial semantics, since the matching algorithm is used only in the backward chaining fragment.

3.3.2 Completeness of the moded semantics

We prove that the moded semantics of Mini-CLF is complete with respect to the oracle semantics, similar to the development of Sect. 2.3 for LF.

The oracle semantics for Mini-CLF is given by the judgments

$$\forall^?\Gamma; \Delta \vdash A$$
$$\forall^?\Gamma_1; \Delta_1 \rightsquigarrow \forall^?\Gamma_2; \Delta_2$$

The former means basically that A is inhabited in $\Gamma; \Delta$, while the latter means that $\forall^?\Gamma_1; \Delta_1$ is transformed into $\forall^?\Gamma_2; \Delta_2$ by a sequence of forward steps. The rules are given in Fig. 3.4. It is easy to see by induction on the rules that if $\forall^?\Gamma_1; \Delta_1 \rightsquigarrow \forall^?\Gamma_2; \Delta_2$, then $\forall^?\Gamma_2$ is of the form $\forall^?\Gamma_1, \forall^?\Gamma_3$ for some $\forall^?\Gamma_3$.

As in the case of LF, the oracle gives the value of ρ in rules osem-atm, osem-mon, and osem-step.

Completeness of the moded semantics with respect to the oracle semantics is expressed similarly to Theorem 2.23.

Theorem 3.4 *Assume that* $\vdash \Sigma$ decl, $\cdot \vdash \Phi$ decl, $\uparrow\Phi \vdash \underline{\forall}\Delta$ decl, $\uparrow\Phi \vdash A$ goal $> \nu$, *and* $\forall^?\Gamma \vdash \rho : \Phi$.

- *If* $\forall^?\Gamma; \rho\Delta \vdash \rho A$, *then there exists* ρ_0 *and* ρ_1 *such that* $\Phi; \Delta \vdash A \Longrightarrow \rho_0; \Phi'$, $\forall^?\Gamma \vdash \rho_1 : \Phi'$ *and* $\rho = \rho_1\rho_0$.
- *If* $\forall^?\Gamma; \rho\Delta_1 \rightsquigarrow \forall^?\Gamma, \forall^?\Gamma_2; \Delta_2$, *then there exists* ρ_0 *and* ρ_1 *such that* $\Phi; \Delta_1 \rightsquigarrow \rho_0; \Phi'; \forall^?\Gamma_2'; \Delta_2'$, $\forall^?\Gamma, \forall^?\Gamma_2 \vdash \rho_1 : \Phi', \forall^?\Gamma_2'$, $\Delta_2 = \rho_1\Delta_2'$ *and* $\rho = \rho_1\rho_0$.

Proof: By induction on the derivation in the oracle semantics and case analysis on the last rule used. We show only the most relevant cases.

osem-mon We have the derivation

$$\frac{\forall^?\Gamma; \rho\Delta \rightsquigarrow \forall^?\Gamma, \forall^?\Gamma_1; \Delta_1 \qquad \forall^?\Gamma, \forall^?\Gamma_1 \vdash \rho_0 : \exists\rho\Gamma_0 \qquad \forall^?\Gamma, \forall^?\Gamma_1; \Delta_1 \vdash \rho_0\rho A_0^+}{\forall^?\Gamma; \rho\Delta \vdash \rho(\{\exists\Gamma_0.A_0^+\})}$$

By IH, there exists ρ_1 and ρ_1' such that $\Phi; \Delta \rightsquigarrow \rho_1; \Phi'; \forall^?\Gamma_1'; \Delta_1'$, $\forall^?\Gamma, \forall^?\Gamma_1 \vdash \rho_1' : \Phi', \forall^?\Gamma_1'$, $\Delta_1 = \rho_1'\Delta_1'$, and $\rho = \rho_1'\rho_1$. Extending with ρ_0 we get $\forall^?\Gamma, \forall^?\Gamma_1 \vdash \rho_1', \rho_0 : \Phi', \forall^?\Gamma_1', \exists\rho_1\Gamma_0$. Note that $\rho_0\rho A_0^+ = \rho_0\rho_1'\rho_1 A_0^+ = (\rho_1', \rho_0)\rho_1 A_0^+$, and $\rho_1'\Delta_1' = (\rho_1', \rho_0)\Delta_1'$. By IH, there exists ρ_2 and ρ_2' such that $\Phi', \forall^?\Gamma_1', \exists\rho_1\Gamma_0; \Delta_1' \vdash \rho_1 A_0^+ \Longrightarrow \rho_2; \Phi_2$, $\forall^?\Gamma, \forall^?\Gamma_1 \vdash \rho_2' : \Phi_2$ and $\rho_1' = \rho_2'\rho_2$. Then, $\rho_2|_{\Phi_2}\rho_1$ and ρ_2' are the desired substitutions.

osem-step We have the derivation

$$\frac{h : \Pi\Gamma_0.A_0^+ \multimap \{\exists\Gamma_1.A_1^+\} \in \Sigma, \Gamma, \rho\Delta \qquad \forall^?\Gamma; (\rho\Delta \setminus\!\setminus h) \vdash \rho_0 A_0^+ \qquad \vec{x}{:}\rho_0 A_1^+ \Longrightarrow \underline{\forall}\Gamma_2; \Delta_2 \qquad \vec{x} \text{ fresh}}{\forall^?\Gamma; \rho\Delta, \rho\Delta_0 \rightsquigarrow \forall^?\Gamma; \forall\rho_0\Gamma_1, \underline{\forall}\Gamma_2; \rho\Delta_0, \Delta_2}$$

By reasoning similar to the case of rule osem-atm in Theorem 2.23, there exists h declared in Σ, Φ, Δ such that h has type $\Pi\Gamma_0'.A_0'^+ \multimap \{\exists\Gamma_1'.A_1'^+\}$ and $\rho(\Pi\Gamma_0'.A_0'^+ \multimap \{\exists\Gamma_1'.A_1'^+\}) = \Pi\Gamma_0.A_0^+ \multimap \{\exists\Gamma_1.A_1^+\}$.

$\boxed{\forall^?\Gamma; \Delta \vdash A^-}$

$$\frac{\vec{x}{:}A_0^+ \Longrightarrow \underline{\forall}\Gamma_1, \Delta_1 \qquad \vec{x} \text{ fresh} \qquad \forall^?\Gamma, \forall\Gamma_0, \underline{\forall}\Gamma_1; \Delta, \Delta_1 \vdash P}{\forall^?\Gamma; \Delta \vdash \Pi\Gamma_0.A_0^+ \multimap P}\ \textsf{osem-type}^-$$

$\boxed{\forall^?\Gamma; \Delta \vdash P}$

$$\frac{\begin{array}{c} h{:}\Pi\Gamma_0.A_0^+ \multimap a{\cdot}S_0 \in \Sigma, \Gamma.\Delta \\ \forall^?\Gamma \vdash \rho : \exists\Gamma_0 \qquad \rho S_0 = S \qquad \forall^?\Gamma; (\Delta \setminus\!\setminus h) \vdash \rho A_0^+ \end{array}}{\forall^?\Gamma; \Delta \vdash a{\cdot}S}\ \textsf{osem-atm}$$

$$\frac{\forall^?\Gamma; \Delta \rightsquigarrow \forall^?\Gamma_1; \Delta_1 \qquad \forall^?\Gamma_1 \vdash \rho : \exists\Gamma_0 \qquad \forall^?\Gamma_1; \Delta_1 \vdash \rho A_0^+}{\forall^?\Gamma; \Delta \vdash \{\exists\Gamma_0.A_0^+\}}\ \textsf{osem-mon}$$

$\boxed{\forall^?\Gamma; \Delta \vdash A^+}$

$$\frac{}{\forall^?\Gamma; \Delta \vdash \mathbf{1}}\ \textsf{osem-one}$$

$$\frac{\forall^?\Gamma; \Delta_1 \vdash A^- \qquad \forall^?\Gamma; \Delta_2 \vdash B^+}{\forall^?\Gamma; \Delta_1, \Delta_2 \vdash {\downarrow}A^- \otimes B^+}\ \textsf{osem-prod-lin}$$

$$\frac{\forall^?\Gamma; \cdot \vdash A^- \qquad \forall^?\Gamma; \Delta \vdash B^+}{\forall^?\Gamma; \Delta \vdash !A^- \otimes B^+}\ \textsf{osem-prod-bang}$$

$\boxed{\forall^?\Gamma; \Delta \rightsquigarrow \forall^?\Gamma'; \Delta'}$

$$\frac{}{\forall^?\Gamma; \Delta \rightsquigarrow \forall^?\Gamma; \Delta}\ \textsf{osem-stop}$$

$$\frac{\begin{array}{c} h : \Pi\Gamma.A^+ \multimap \{\exists\Gamma_0.A_0^+\} \in \Sigma, \Gamma, \Delta \qquad \forall^?\Gamma; (\Delta \setminus\!\setminus h) \vdash \rho A^+ \\ \vec{x}{:}\rho A_0^+ \Longrightarrow \underline{\forall}\Gamma_1; \Delta_1 \qquad \vec{x} \text{ fresh} \end{array}}{\forall^?\Gamma; \Delta, \Delta_0 \rightsquigarrow \forall^?\Gamma, \forall\rho\Gamma_0, \underline{\forall}\Gamma_1; \Delta_0, \Delta_1}\ \textsf{osem-step}$$

$$\frac{\forall^?\Gamma_0; \Delta_0 \rightsquigarrow \forall^?\Gamma_1; \Delta_1 \qquad \forall^?\Gamma_1; \Delta_1 \rightsquigarrow \forall^?\Gamma_2; \Delta_2}{\forall^?\Gamma_0; \Delta_0 \rightsquigarrow \forall^?\Gamma_2; \Delta_2}\ \textsf{osem-comp}$$

Figure 3.4: Oracle semantics for Mini-CLF

Since $\forall^?\Gamma \vdash \rho_0 : \exists\Gamma_0$, and $\forall^?\Gamma \vdash \rho : \Phi$, we have $\forall^?\Gamma \vdash \rho, \rho_0\rho : \Phi, \exists\Gamma'_0$. By IH, there exists ρ_1 and ρ'_1 such that $\Phi, \exists\Gamma'_0; \Delta \vdash {A_0}'^+ \Longrightarrow \rho_1; \Phi_1$, $\forall^?\Gamma \vdash \rho'_1 : \Phi_1$ and $\rho, \rho_0\rho = \rho'_1\rho_1$. From the latter, $\rho = \rho'_1\rho_1|_\Phi$.

On Γ'_0, $\rho_0\rho$ coincide with $\rho'_1\rho_1$, then $\rho_0 A_1^+ = \rho_0\rho {A_1}'^+ = \rho'_1\rho_1 {A_1}'^+$. Then, there exists Γ'_2 and Δ'_2 such that $\vec{x}{:}\rho_1 {A_1}'^+ \Longrightarrow \Gamma'_2; \Delta'_2$.

Then, $\rho_1|_\Phi$, with $\Phi_1 \vdash \rho_1|_\Phi : \Phi$, and ρ'_1, with $\forall^?\Gamma, \underline{\forall}\Gamma_2 \vdash \rho'_1 : \Phi_1, \underline{\forall}\Gamma'_2$, satisfy the required conditions.

□

A Mode Checking Rules for CLF

In this section, we define the mode checker for the full language of CLF, as it is implemented in `Celf` [15].

A.1 Types and Terms

$$
\begin{array}{rll}
K & ::= \mathsf{type} \mid \Pi x{:}A^{-}.K & \text{(Kinds)} \\
A^{-} & ::= a{\cdot}S' \mid A_1^{-} \;\&\; A_2^{-} \mid \Pi p{:}A^{+}.A^{-} \mid A^{+} & \text{(Negative types)} \\
A^{+} & ::= \mathbf{1} \mid \exists p{:}A^{+}.A^{+} \mid\; !A^{-} \mid\; @A^{-} \mid\; {\downarrow}A^{-} & \text{(Positive types)} \\
N & ::= H{\cdot}S \mid \langle N_1, N_2\rangle \mid \widehat{\lambda} p.N \mid E & \text{(Negative terms)} \\
H & ::= x \mid c & \text{(Heads)} \\
S' & ::= () \mid N; S & \text{(Type spines)} \\
S & ::= () \mid \pi_1; S \mid \pi_2; S \mid M; S & \text{(Spines)} \\
E & ::= M \mid \mathsf{let}\ \epsilon\ \mathsf{in}\ E & \text{(Expressions)} \\
\epsilon & ::= \cdot \mid \delta \mid \epsilon_1; \epsilon_2 & \text{(Traces)} \\
\delta & ::= \{p\} = H{\cdot}S & \text{(Steps)} \\
p & ::= 1 \mid \langle\!\langle p_1, p_2, |\rangle\!\rangle !x \mid @x \mid {\downarrow}x & \text{(Patterns)} \\
M & ::= 1 \mid \langle\!\langle M_1, M_2, |\rangle\!\rangle !N \mid @N \mid {\downarrow}N & \text{(Positive terms)} \\
\Sigma & ::= \cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A^{-} & \text{(Signatures)}
\end{array}
$$

Note that the spines S' in atomic types have the form $S' ::= () \mid M; S'$ (because type families are classified by kinds, which do not admit & as a constructor).

For the purpose of mode-checking, the distinction between dependent and non-dependent types or variables is essential, while the differences between persistent, affine and linear types or variable is irrelevant. We will then write $\Box$ for any one of the modalities !, @ and $\downarrow$ (e.g., in $\Box A^{-}$).

A.2 Normal Patterns

The general definition of CLF patterns leads to unwieldy mode rules. For this reason, we consider *normal patterns*, which essentially reassociate general patterns to the right. There is nothing special about normal patterns, except that they simplify many of the rules in this report.

$$\text{Normal patterns} \qquad np \;::=\; 1 \mid \langle\!\langle !x, np\rangle\!\rangle \mid \langle\!\langle @x, np\rangle\!\rangle \mid \langle\!\langle {\downarrow}x, np\rangle\!\rangle$$

Normal patterns have the form: $\langle\!\langle \Box x_1, \langle\!\langle \Box x_2, \ldots \langle\!\langle \Box x_n, 1\rangle\!\rangle \ldots\rangle\!\rangle\rangle\!\rangle$, where $\Box x_i$ is one of $!x_i$, $@x_i$ or ${\downarrow}x_i$.

Pattern Normalization $\boxed{p : A^{+} \mapsto np : B^{+}}$
Functional reading: $(p, A^{+}) \mapsto (np, B^{+})$
Defined inductively on $p ::= 1 \mid \langle\!\langle p_1, p_2\rangle\!\rangle \mid \Box x$

Generic patterns (and their type) can be compiled into normal patterns by means of the rules below:

$$\frac{}{1 : \mathbf{1} \mapsto 1 : \mathbf{1}}\ \textsf{pn-1}$$

$$\frac{p : A^+ \mapsto np : B^+}{\langle\langle 1, p\rangle\rangle : \exists 1{:}\mathbf{1}.\, A^+ \mapsto np : B^+}\ \textsf{pn-1-ex}$$

$$\frac{\langle\langle p_1, \langle\langle p_2, p_3\rangle\rangle\rangle\rangle : \exists p_1{:}A_1^+.\, (\exists p_2{:}A_2^+.\, A_3^+) \mapsto np : B^+}{\langle\langle\langle\langle p_1, p_2\rangle\rangle, p_3\rangle\rangle : \exists \langle\langle p_1, p_2\rangle\rangle{:}(\exists p_1{:}A_1^+.\, A_2^+).\, A_3^+ \mapsto np : B^+}\ \textsf{pn-ex-ex}$$

$$\frac{}{\Box x : \Box A^- \mapsto \langle\langle \Box x, 1\rangle\rangle : \exists \Box x{:}\Box A^-.\, \mathbf{1}}\ \textsf{pn-}\Box$$

$$\frac{p : A_2^+ \mapsto np : B^+}{\langle\langle \Box x, p\rangle\rangle : \exists \Box x{:}\Box A_1^-.\, A_2^+ \mapsto \langle\langle \Box x, np\rangle\rangle : \exists \Box x{:}\Box A_1^-.\, B^+}\ \textsf{pn-}\Box\textsf{-ex}$$

The type B^+ differs from A^+ by the reassociation of inner existentials to the right and by the possible addition of 1 at the end.

A.3 Mode Checking

Abstract substitutions Recall that the mode checker produces an abstract substitution representing the groundness information of the dependent arguments of a context. For readability, the abstract substitutions we use in this section do not contain types. We thus define abstract substitutions as follows:

$$\nu ::= \cdot \mid \nu, \forall x, \mid \nu, \exists^{\mathsf{u}} x \mid \nu, \exists^{\mathsf{g}} x$$

We use the following meet-like and join-like operations on abstract substitutions, denoted $\nu_1 \sqcup \nu_2$ and $\nu_1 \sqcap \nu_2$, defined as the point-wise extension of the following operations on variable declarations:

$$\begin{array}{rlrl}
\forall x \sqcup \forall x &= \forall x & \forall x \sqcap \forall x &= \forall x \\
\exists^{?} x \sqcup \exists^{\mathsf{g}} x &= \exists^{\mathsf{g}} x & \exists^{?} x \sqcap \exists^{\mathsf{u}} x &= \exists^{\mathsf{u}} x \\
\exists^{\mathsf{g}} x \sqcup \exists^{?} x &= \exists^{\mathsf{g}} x & \exists^{\mathsf{u}} x \sqcap \exists^{?} x &= \exists^{\mathsf{u}} x \\
\exists^{\mathsf{u}} x \sqcup \exists^{\mathsf{u}} x &= \exists^{\mathsf{u}} x & \exists^{\mathsf{g}} x \sqcap \exists^{\mathsf{g}} x &= \exists^{\mathsf{g}} x
\end{array}$$

Mode checking It is defined by the following judgments:

$\vdash \Sigma$ prog	(mode-checking a logic program)
$\nu \vdash A^-$ decl	(mode-checking a declaration)
A^- fchain	(forward chaining declaration)
A^- bchain	(backward chaining declaration)
$\nu \vdash A^-$ fwd	(mode-checking a fw-chaining declaration)
$\nu \vdash p : A_1^+ [A_2^-]$ fwd	(mode-checking fw-chaining dependent type)
$\nu \vdash A^+$ fwd	(mode-checking a fw-chaining monad)
$\nu \vdash p : A_1^+ [A_2^+]$ fwd	(mode-checking a fw-chaining exists type)
$\Gamma \Vdash p$	(extracting pattern variables)
$\nu \vdash A^-$ bwd $> \nu'$	(mode-checking bw-chaining programs)
$\nu \vdash S : m$ head $> \nu'$	(mode-checking a head)
$\nu \vdash p : A_1^+ [A_2^-]$ bwd $> \nu'$	(extracting patterns in bw search)
$\nu \vdash A^-$ goal $> \nu'$	(mode-checking negative goals)
$\nu \vdash S : m$ goal $> \nu'$	(mode-checking atomic negative goals)
$\nu \vdash A^+$ goal $> \nu'$	(mode-checking positive goals)
$\nu \vdash p : A_1^+ [A_2^+]$ goal $> \nu'$	(mode-checking exists goals)
$\nu \vdash p : A_1^+ [A_2^-]$ goal $> \nu'$	(mode-checking goal patterns)
$\nu \vdash M > \nu'$	(infer groundness for monadic terms)
$\nu \vdash M$	(check groundness for monadic terms)
$\nu \vdash^{\mathsf{G}} M > \nu'$	(request groundness for monadic terms)

In all judgments, we assume to be working with well typed objects with respect to some larger CLF specification, which also includes kind declarations for all type families, and type declarations for all constants that are not interpreted operationally as (logic) programs. For the sake of readability, we do not mention this CLF specification in the above judgments as it stays fixed.

We also assume that this specification contains mode declarations for the type families that are to be interpreted operationally. A mode declaration for a type family a is a sequence of modes whose length is the number of arguments of a; we denote it with $\mathsf{mode}(a)$.

A.4 Programs

Judgments:

$\vdash \Sigma$ prog	(mode-checking a logic program)
$\nu \vdash A^-$ decl	(mode-checking a declaration)
A^- fchain	(forward chaining constant)
A^- bchain	(backward chaining constant)

A.4.1 Programs $\boxed{\vdash \Sigma \text{ prog}}$

Functional reading: $\Sigma \mapsto \{true, false\}$
Defined inductively on $\Sigma ::= \cdot \mid \Sigma, c : A^-$

$$\frac{}{\vdash \cdot \ \mathsf{prog}}\ \mathsf{sig\text{-}empty} \qquad \frac{\vdash \Sigma \ \mathsf{prog} \quad \cdot \vdash A^- \ \mathsf{decl}}{\vdash \Sigma, c : A^- \ \mathsf{prog}}\ \mathsf{sig\text{-}decl}$$

We remark that programs are always negative, and hence, we only need one judgment for checking negative types as programs.

A.4.2 Declarations $\boxed{\nu \vdash A^- \ \mathsf{decl}}$

Functional reading: $(\nu, A^-) \mapsto \{true, false\}$
Defined based on whether A^- is a forward- or backward-chaining definition.

$$\frac{A^- \ \mathsf{fchain} \quad \nu \vdash A^- \ \mathsf{fwd}}{\nu \vdash A^- \ \mathsf{decl}}\ \mathsf{prog\text{-}fwd}$$

$$\frac{A^- \ \mathsf{bchain} \quad \nu \vdash A^- \ \mathsf{bwd} \ > \nu'}{\nu \vdash A^- \ \mathsf{decl}}\ \mathsf{prog\text{-}bwd}$$

A.4.3 Mode Direction $\boxed{A^- \ \mathsf{fchain}}$ $\boxed{A^- \ \mathsf{bchain}}$

A declaration $c : A^-$ may have multiple heads when A^- contains additive conjunctions. If a head is monadic, then the constant is used in forward chaining fashion, if it is not, it is a backward chaining constant. We require that all heads of a declaration be either one or the other.

For the definition, we use some auxiliary judgments:

- $A^+ \ \mathsf{okchain}$ checks that embedded clauses in A^+ are either backward chaining or forward chaining;
- $p : A^+ \ \mathsf{okchain}$ checks that non-dependent clauses in A^+ satisfy the $\mathsf{okchain}$ predicate;
- $A^- \ \mathsf{okgoal}$ checks that embedded clauses in A^- satisfy the $\mathsf{okchain}$ predicate; $A^+ \ \mathsf{okgoal}$ is also defined with similar meanings.

$\boxed{A^- \ \mathsf{bchain}}$

$$\frac{}{P \ \mathsf{bchain}} \qquad \frac{A_1^- \ \mathsf{bchain} \quad A_2^- \ \mathsf{bchain}}{A_1^- \ \& \ A_2^- \ \mathsf{bchain}} \qquad \frac{A_1^+ \ \mathsf{okgoal} \quad A_2^- \ \mathsf{bchain}}{\Pi p{:}A_1^+ . \, A_2^- \ \mathsf{bchain}}$$

$\boxed{A^- \ \mathsf{fchain}}$

$$\frac{A^+ \ \mathsf{okchain}}{\{A^+\} \ \mathsf{fchain}} \qquad \frac{A_1^- \ \mathsf{fchain} \quad A_2^- \ \mathsf{fchain}}{A_1^- \ \& \ A_2^- \ \mathsf{fchain}} \qquad \frac{A_1^+ \ \mathsf{okgoal} \quad A_2^- \ \mathsf{fchain}}{\Pi p{:}A_1^+ . \, A_2^- \ \mathsf{fchain}}$$

$\boxed{A^- \ \mathsf{okgoal}}$

$$\frac{}{P \ \mathsf{okgoal}} \qquad \frac{A^+ \ \mathsf{okgoal}}{\{A^+\} \ \mathsf{okgoal}} \qquad \frac{A_1^- \ \mathsf{okgoal} \quad A_2^- \ \mathsf{okgoal}}{A_1^- \ \& \ A_2^- \ \mathsf{okgoal}} \qquad \frac{A_1^+ \ \mathsf{okchain} \quad A_2^- \ \mathsf{okgoal}}{\Pi p{:}A_1^+ . \, A_2^- \ \mathsf{okgoal}}$$

$\boxed{A^+ \text{ okgoal}}$

$$\frac{}{\mathbf{1} \text{ okgoal}} \qquad \frac{x \in FV(A_2^+) \quad A_2^+ \text{ okgoal}}{\exists !x{:}!A_1^+ . A_2^+ \text{ okgoal}} \qquad \frac{x \notin FV(A_1^+) \quad A_1^- \text{ okgoal} \quad A_2^+ \text{ okgoal}}{\exists \Box x{:}\Box A_1^+ . A_2^+ \text{ okgoal}}$$

$\boxed{A^- \text{ okchain}}$

$$\frac{A^- \text{ bchain}}{A^- \text{ okchain}} \qquad \frac{A^- \text{ fchain}}{A^- \text{ okchain}}$$

$\boxed{A^+ \text{ okchain}}$

$$\frac{}{\mathbf{1} \text{ okchain}} \qquad \frac{x \in FV(A_2^+) \quad A_2^+ \text{ okchain}}{\exists !x{:}!A_1^+ . A_2^+ \text{ okchain}} \qquad \frac{x \notin FV(A_2^+) \quad A_1^- \text{ okchain} \quad A_2^+ \text{ okchain}}{\exists \Box x{:}\Box A_1^+ . A_2^+ \text{ okchain}}$$

A.5 Declarations in Forward Chaining Mode

Judgments:

$\nu \vdash A^- \text{ fwd}$	(mode-checking a fw-chaining declaration)
$\nu \vdash p : A_1^+[A_2^-] \text{ fwd}$	(mode-checking fw-chaining dependent type)
$\nu \vdash A^+ \text{ fwd}$	(mode-checking a fw-chaining monad)
$\Gamma \Vdash p$	(extracting pattern variables)

A.5.1 Declarations $\boxed{\nu \vdash A^- \text{ fwd}}$

Functional reading: $(\nu, A^-) \mapsto \{true, false\}$
Defined inductively on $A^- ::= a{\cdot}S \mid A_1^- \;\&\; A_2^- \mid \Pi p{:}A^+ . A^- \mid \{A^+\}$

$$\text{(no rule for } \nu \vdash a{\cdot}S \text{ fwd)}$$

$$\frac{\nu \vdash A_1^- \text{ fwd} \quad \nu \vdash A_2^- \text{ fwd}}{\nu \vdash A_1^- \;\&\; A_2^- \text{ fwd}} \text{ fwd-and}$$

$$\frac{\nu \vdash p : A_1^+[A_2^-] \text{ fwd}}{\nu \vdash \Pi p{:}A_1^+ . A_2^- \text{ fwd}} \text{ fwd-pi}$$

$$\frac{\nu \vdash A^+ \text{ fwd}}{\nu \vdash \{A^+\} \text{ fwd}} \text{ fwd-mon}$$

A.5.2 Left-Hand Side of a Monad $\boxed{\nu \vdash np : A^+[B^-]\ \mathsf{fwd}}$

Functional reading: $(\nu, np, A^+, B^-) \mapsto \{true, false\}$
Defined inductively on $np ::= 1 \mid \langle\!\langle !x, np \rangle\!\rangle \mid \langle\!\langle \Box x, np \rangle\!\rangle$

$$\frac{\nu \vdash B^-\ \mathsf{fwd}}{\nu \vdash 1 : \mathbf{1}[B^-]\ \mathsf{fwd}}\ \mathsf{fw\text{-}lhs\text{-}1}$$

$$\frac{\nu, \exists^{\mathsf{u}} x \vdash np : A_2^+[B^-]\ \mathsf{fwd} \qquad x \in FV(A_2^+, B^-)}{\nu \vdash \langle\!\langle !x, np \rangle\!\rangle : (\exists !x{:}!A_1^-.\, A_2^+)[B^-]\ \mathsf{fwd}}\ \mathsf{fw\text{-}lhs\text{-}bang}$$

$$\frac{\nu \vdash A_1^-\ \mathsf{goal}\ > \nu' \quad \nu' \vdash np : A_2^+[B^-]\ \mathsf{fwd}}{\nu \vdash \langle\!\langle \Box x, np \rangle\!\rangle : (\exists \Box x{:}\Box A_1^-.\, A_2^+)[B^-]\ \mathsf{fwd}}\ \mathsf{fw\text{-}\Box}$$

A.5.3 Monads $\boxed{\nu \vdash A^+\ \mathsf{fwd}}$

Functional reading: $(\nu, A^+) \mapsto \{true, false\}$
Defined inductively on $A^+ ::= \mathbf{1} \mid \exists p{:}A_1^+.\, A_2^+ \mid \Box A^-$

$$\frac{}{\nu \vdash \mathbf{1}\ \mathsf{fwd}}\ \mathsf{fw\text{-}mon\text{-}one}$$

$$\frac{\nu \vdash p : A_1^+[A_2^+]\ \mathsf{fwd}}{\nu \vdash \exists p{:}A_1^+.\, A_2^+\ \mathsf{fwd}}\ \mathsf{fw\text{-}mon\text{-}exists}$$

$$\frac{\nu \vdash A^-\ \mathsf{decl}}{\nu \vdash \Box A^-\ \mathsf{fwd}}\ \mathsf{fw\text{-}\Box}$$

A.5.4 Monads $\boxed{\nu \vdash p : A_1^+[A_2^+]\ \mathsf{fwd}}$

Functional reading: $(\nu, p, A_1^+, A_2^+) \mapsto \{true, false\}$
Defined inductively on $p ::= 1 \mid \langle\!\langle \Box x, np \rangle\!\rangle$

$$\frac{\nu \vdash A^+\ \mathsf{fwd}}{\nu \vdash 1 : \mathbf{1}[A^+]\ \mathsf{fwd}}\ \mathsf{fw\text{-}exists\text{-}one}$$

$$\frac{\nu, \forall x \vdash np : A_2^+[B^+]\ \mathsf{fwd} \qquad x \in FV(A_2^+, B^+)}{\nu \vdash \langle\!\langle !x, np \rangle\!\rangle : \exists !x{:}A_1^+.\, A_2^+[B^+]\ \mathsf{fwd}}\ \mathsf{fw\text{-}exists}$$

$$\frac{\nu \vdash A_1^+\ \mathsf{fwd} \qquad \nu, \forall x \vdash np : A_2^+[B^+]\ \mathsf{fwd} \qquad x \notin FV(A_2^+, B^+)}{\nu \vdash \langle\!\langle \Box x, np \rangle\!\rangle : \exists \Box x{:}A_1^+.\, A_2^+[B^+]\ \mathsf{fwd}}\ \mathsf{fw\text{-}exists}$$

A.5.5 Extracting Pattern Variables $\boxed{\Gamma \Vdash p}$

Functional reading: $np \mapsto \Gamma$
Defined inductively on $np ::= 1 \mid \langle\langle \Box x, np \rangle\rangle$

$$\frac{}{\cdot \Vdash 1}\ \textsf{pattern-one}$$

$$\frac{\Gamma \Vdash np}{x, \Gamma \Vdash \langle\langle \Box x, np \rangle\rangle}\ \textsf{pattern-}\Box$$

A.6 Declarations in Backward Chaining Mode

Judgments:

$\nu \vdash A^- \ \textsf{bwd} \ > \nu'$	(mode-checking bw-chaining programs)
$\nu \vdash S : m \ \ \textsf{head} \ > \nu'$	(mode-checking a head)
$\nu \vdash p : A_1^+[A_2^-] \ \textsf{bwd} \ > \nu'$	(extracting patterns in bw search)
$\nu \vdash A^- \ \textsf{goal} \ > \nu'$	(mode-checking negative goals)
$\nu \vdash S : m \ \ \textsf{goal} \ > \nu'$	(mode-checking atomic negative goals)
$\nu \vdash A^+ \ \textsf{goal} \ > \nu'$	(mode-checking positive goals)
$\nu \vdash p : A_1^+[A_2^-] \ \textsf{goal} \ > \nu'$	(mode-checking goal patterns)

Backward chaining rules make use of groundness obligations. Recall that an entire declaration must be in backward chaining, not just part of it.

A.6.1 Declarations $\boxed{\nu \vdash A^- \ \textsf{bwd} \ > \nu'}$

Functional reading: $(\nu, A^-) \mapsto \nu'$
Defined inductively on $A^- ::= P \mid A_1^- \ \& \ A_2^- \mid \Pi p{:}A^+ . \, A^- \mid \{A^+\}$

$$\frac{\nu \vdash S : \textsf{mode}(a) \ \ \textsf{head} \ > \nu'}{\nu \vdash (a \cdot S)^- \ \textsf{bwd} \ > \nu'}\ \textsf{bwd-head}$$

$$\frac{\nu \vdash A_1^- \ \textsf{bwd} \ > \nu_1 \quad \nu \vdash A_2^- \ \textsf{bwd} \ > \nu_2}{\nu \vdash A_1^- \ \& \ A_2^- \ \textsf{bwd} \ > \nu_1 \sqcup \nu_2}\ \textsf{bwd-and}$$

$$\frac{\nu \vdash p : A_1^+[A_2^-] \ \textsf{bwd} \ > \nu'}{\nu \vdash \Pi p{:}A_1^+ . \, A_2^- \ \textsf{bwd} \ > \nu'}\ \textsf{bwd-pi}$$

(no rule for $\nu \vdash \{A^+\} \ \textsf{bwd} \ > \nu'$)

$\textsf{mode}(a)$ is the declared mode for type family a. In rule bwd-and one among the left or right conjunct will be executed, but we do not know which one, so both must enforce groundness.

A.6.2 Heads $\boxed{\nu \vdash S : m \ \mathsf{head} \ > \nu'}$

Functional reading: $(\nu, S, m) \mapsto \nu'$
Defined inductively on $m ::= () \mid + \to m \mid - \to m$

$$\frac{}{\nu \vdash () : () \ \mathsf{head} \ > \nu} \ \textsf{bw-head-nil}$$

$$\frac{\nu \vdash N > \nu' \quad \nu \vdash S : m \ \mathsf{head} \ > \nu''}{\nu \vdash (N; S) : (+ \to m) \ \mathsf{head} \ > \nu' \sqcap \nu''} \ \textsf{bw-head-input}$$

$$\frac{\nu \vdash S : m \ \mathsf{head} \ > \nu''}{\nu \vdash (N; S) : (- \to m) \ \mathsf{head} \ > \nu' \sqcap \nu''} \ \textsf{bw-head-output}$$

We infer that all variables in input position are ground. Variables occurring in output positions are checked to be ground after processing the goals (see below). There is no reason to sequentialize the manipulation of variables.

A.6.3 Declaration Body $\boxed{\nu \vdash p : A^+[B^-] \ \mathsf{bwd} \ > \nu'}$

Functional reading: $(\nu, np, A^+, B^-) \mapsto \nu'$
Defined inductively on $np ::= 1 \mid \langle\!\langle !x, np \rangle\!\rangle \mid \langle\!\langle \Box x, np \rangle\!\rangle$

$$\frac{\nu \vdash B^- \ \mathsf{bwd} \ > \nu'}{\nu \vdash 1 : \mathbf{1}[B^-] \ \mathsf{bwd} \ > \nu'} \ \textsf{bw-body-1}$$

$$\frac{\nu, \exists^{\mathsf{u}} x \vdash np : A_2^+[B^-] \ \mathsf{bwd} \ > \nu', \exists^{\mathsf{g}} x \qquad x \in FV(A_2^+, B^-)}{\nu \vdash \langle\!\langle !x, np \rangle\!\rangle : (\exists !x{:}!A_1^- \,.\, A_2^+)[B^-] \ \mathsf{bwd} \ > \nu'} \ \textsf{bw-body-bang}$$

$$\frac{\nu' \vdash A_1^- \ \mathsf{goal} \ > \nu'' \qquad \nu, \forall x \vdash np : A_2^+[B^-] \ \mathsf{bwd} \ > \nu', \forall x \qquad x \notin FV(A_2^+, B^-)}{\nu \vdash \langle\!\langle \Box x, np \rangle\!\rangle : (\exists \Box x{:}\Box A_1^- \,.\, A_2^+)[B^-] \ \mathsf{bwd} \ > \nu''} \ \textsf{bw-body-}\Box$$

Rule bw-body-bang states that dependent arguments must be made ground after processing goals. Input arguments are made ground when processing the head, while output arguments should be made ground when processing the goals. If this rule does not apply (i.e., the variable x is not made ground), then mode checking fails.

A.6.4 Negative Goals $\boxed{\nu \vdash A^- \ \mathsf{goal} \ > \nu'}$

Functional reading: $(\nu, A^-) \mapsto \nu'$
Defined inductively on $A^- ::= P \mid A_1^- \ \& \ A_2^- \mid \Pi p{:}A^+ \,.\, A^- \mid \{A^+\}$

$$\frac{\nu \vdash S : \mathsf{mode}(a) \ \mathsf{goal} \ > \nu'}{\nu \vdash (a \cdot S)^- \ \mathsf{goal} \ > \nu'} \ \textsf{goal-atomic}$$

$$\frac{\nu \vdash A_1^- \ \mathsf{goal} \ > \nu_1 \quad \nu_1 \vdash A_2^- \ \mathsf{goal} \ > \nu_2}{\nu \vdash A_1^- \ \& \ A_2^- \ \mathsf{goal} \ > \nu_2} \ \textsf{goal-and}$$

$$\frac{\nu \vdash p : A_1^+[A_2^-] \ \mathsf{goal} \ > \nu'}{\nu \vdash \Pi p{:}A_1^+ . \, A_2^- \ \mathsf{goal} \ > \nu'} \ \mathsf{goal\text{-}pi}$$

$$\frac{\nu \vdash A^+ \ \mathsf{goal} \ > \nu'}{\nu \vdash \{A^+\} \ \mathsf{goal} \ > \nu'} \ \mathsf{goal\text{-}mon}$$

$\mathsf{mode}(a)$ is the declared mode for type family a.

In rule goal-and, the subgoals A_1^- and A_2^- are independent, and yet their processing is sequentialized. This is because both must be evaluated, and the operational semantics of Celf (and LLF) processes them left-to-right.

A.6.5 Atomic Goals $\boxed{\nu \vdash S : m \ \mathsf{goal} \ > \nu'}$

Functional reading: $(\nu, S, m) \mapsto \nu'$
Defined inductively on $m ::= () \mid + \to m \mid - \to m$

$$\frac{}{\nu \vdash () : () \ \mathsf{goal} \ > \nu} \ \mathsf{goal\text{-}nil}$$

$$\frac{\nu \vdash M \quad \nu \vdash S : m \ \mathsf{goal} \ > \nu'}{\nu \vdash (M; S) : (+ \to m) \ \mathsf{goal} \ > \nu'} \ \mathsf{goal\text{-}input}$$

$$\frac{\nu \vdash M > \nu' \quad \nu \vdash S : m \ \mathsf{goal} \ > \nu''}{\nu \vdash (M; S) : (- \to m) \ \mathsf{goal} \ > \nu' \sqcap \nu''} \ \mathsf{goal\text{-}output}$$

We check that all variables in input positions are ground and assume that the variables in output position are ground for mode-checking the rest of the clause. We do not need to sequentialize the processing of the arguments.

A.6.6 Monadic Goals $\boxed{\nu \vdash A^+ \ \mathsf{goal} \ > \nu'}$

Functional reading: $(\nu, A^+) \mapsto \nu'$
Defined inductively on $A^+ ::= \mathbf{1} \mid \exists p{:}A_1^+ . \, A_2^+ \mid \Box A^-$

$$\frac{}{\nu \vdash \mathbf{1} \ \mathsf{goal} \ > \nu} \ \mathsf{goal\text{-}one}$$

$$\frac{\nu \vdash p : A_1^+[A_2^+] \ \mathsf{goal} \ > \nu'}{\nu \vdash \exists p{:}A_1^+ . \, A_2^+ \ \mathsf{goal} \ > \nu'} \ \mathsf{goal\text{-}exists}$$

$$\frac{\nu \vdash A^- \ \mathsf{goal} \ > \nu'}{\nu \vdash \Box A^- \ \mathsf{goal} \ > \nu'} \ \mathsf{goal\text{-}\Box}$$

A.6.7 Monadic Exist Goals $\boxed{\nu \vdash p : A_1^+[A_2^+]\ \mathsf{goal}\ > \nu'}$

Functional reading: $(\nu, p, A_1^+, A_2^+) \mapsto \nu'$
Defined inductively on $np ::= 1 \mid \langle\langle \Box x, np \rangle\rangle$

$$\frac{\nu \vdash A^+\ \mathsf{goal}\ > \nu'}{\nu \vdash 1 : \mathbf{1}[A^+]\ \mathsf{goal}\ > \nu'}\ \mathsf{exists\text{-}goal\text{-}one}$$

$$\frac{\nu, \exists^{\mathsf{u}} x \vdash np : A_2^+[B^+]\ \mathsf{goal}\ > \nu', \exists^{\mathsf{g}} x \qquad x \in FV(A_2^+, B^+)}{\nu \vdash \langle\langle !x, np \rangle\rangle : (\exists !x{:}!A_1^-\,.\,A_2^+)[B^+]\ \mathsf{goal}\ > \nu'}\ \mathsf{exists\text{-}goal\text{-}bang}$$

$$\frac{\nu \vdash A_1^-\ \mathsf{goal}\ > \nu' \quad \nu', \forall x \vdash np : A_2^+[B^+]\ \mathsf{goal}\ > \nu'', \forall x \qquad x \notin FV(A_2^+, B^+)}{\nu \vdash \langle\langle \Box x, np \rangle\rangle : (\exists \Box x{:}\Box A_1^-\,.\,A_2^+)[B^+]\ \mathsf{goal}\ > \nu''}\ \mathsf{exists\text{-}goal\text{-}\Box}$$

A.6.8 Embedded Clauses $\boxed{\nu \vdash p : A_1^+[A_2^-]\ \mathsf{goal}\ > \nu'}$

Functional reading: $(\nu, np, A^+, B^-) \mapsto \nu'$
Defined inductively on $np ::= 1 \mid \langle\langle !x, np \rangle\rangle \mid \langle\langle \Box x, np \rangle\rangle$

$$\frac{\nu \vdash B^-\ \mathsf{goal}\ > \nu'}{\nu \vdash 1 : \mathbf{1}[B^-]\ \mathsf{goal}\ > \nu'}\ \mathsf{bw\text{-}emb\text{-}1}$$

$$\frac{\nu, \forall x \vdash np : A_2^+[B^-]\ \mathsf{goal}\ > \nu', \forall x \qquad x \in FV(A_2^+, B^-)}{\nu \vdash \langle\langle !x, np \rangle\rangle : (\exists !x{:}!A_1^-\,.\,A_2^+)[B^-]\ \mathsf{goal}\ > \nu'}\ \mathsf{bw\text{-}emb\text{-}bang}$$

$$\frac{\nu \vdash A_1^-\ \mathsf{decl} \qquad \nu, \forall x \vdash np : A_2^+[B^-]\ \mathsf{goal}\ > \nu', \forall x \qquad x \notin FV(A_2^+, B^-)}{\nu \vdash \langle\langle \Box x, np \rangle\rangle : (\exists \Box x{:}\Box A_1^-\,.\,A_2^+)[B^-]\ \mathsf{goal}\ > \nu'}\ \mathsf{bw\text{-}emb\text{-}\Box}$$

A.7 Term judgments

Let us recall the mode checking judgments for terms:

$$\begin{array}{ll} \nu \vdash N > \nu' & \text{(infer groundness for negative terms)} \\ \nu \vdash N & \text{(check groundness for negative terms)} \end{array}$$

The judgment $\nu \vdash T > \nu'$, assume that M term M is ground, and convert any free variable in M of unknown status within ν into a ground variable in ν', with the condition that M is a pattern. A term M is a pattern if its free existential variables are applied to distinct bound variables I.e., if $X \in FV(T)$, then it occurs in terms of the form $X \cdot x_1; \ldots; x_p; ()$ where $x_1, \ldots, x_p$ are distinct bound variables. See [16, 14] for the concrete definition of pattern for CLF, which is an extension of the pattern fragment defined by Miller [9]. For expressions, we require that there is at most one unknown variable of monadic type [2, 3]. Concretely, the judgment $\nu \vdash M > \nu'$, is specified as follows:

$$\nu \vdash M > \nu' \quad \text{iff} \quad M \text{ is a pattern and for all } x \in FV(T), \text{ if } \exists^{\mathsf{u}} x \in \nu \text{ then } \exists^{\mathsf{g}} x \in \nu'$$

The judgment $\nu \vdash T$ checks that all free variables of term M are either universal or ground in ν. Concretely,

$$\nu \vdash T \quad \text{iff} \quad \text{for all } x \in FV(T), \text{ either } \exists^{\mathsf{g}} x \in \nu \text{ or } \forall x \in \nu$$

(but not $\exists^{\mathsf{u}} x \in \nu$).

References

[1] Iliano Cervesato and Frank Pfenning. A linear spine calculus. *Journal of Logic and Computation*, 13(5):639–688, 2003.

[2] Iliano Cervesato, Frank Pfenning, Jorge Luis Sacchini, Carsten Schürmann, and Robert J. Simmons. On Matching in CLF. Technical Report CMU-CS-12-114, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, July 2012.

[3] Iliano Cervesato, Frank Pfenning, Jorge Luis Sacchini, Carsten Schürmann, and Robert J. Simmons. Trace Matching in a Concurrent Logical Framework. In Adam Chlipala and Carsten Schürmann, editors, *7th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice — LFMTP'12*, Copenhagen, Denmark, September 2012.

[4] Iliano Cervesato, Frank Pfenning, David Walker, and Kevin Watkins. A Concurrent Logical Framework II: Examples and Applications. Technical Report CMU-CS-02-102, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 2003.

[5] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[6] Robert Harper and Daniel R. Licata. Mechanizing metatheory in a logical framework. *J. Funct. Program.*, 17(4-5):613–673, July 2007.

[7] Pablo López, Frank Pfenning, Jeff Polakow, and Kevin Watkins. Monadic concurrent linear logic programming. In Pedro Barahona and Amy P. Felty, editors, *PPDP*, pages 35–46. PUB-ACM, 2005.

[8] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[9] Dale Miller. Unification of simply typed lamda-terms as logic programming. In Koichi Furukawa, editor, *ICLP*, pages 255–269. MIT Press, 1991.

[10] Frank Pfenning and Carsten Schürmann. System description: Twelf - A meta-logical framework for deductive systems. In Harald Ganzinger, editor, *CADE*, volume 1632 of *LNCS*, pages 202–206. Springer, 1999.

[11] Ekkehard Rohwedder and Frank Pfenning. Mode and termination checking for higher-order logic programs. In Hanne Riis Nielson, editor, *ESOP*, volume 1058 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 1996.

[12] Jorge Luis Sacchini, Iliano Cervesato, Frank Pfenning, Carsten Schürmann, and Robert J. Simmons. On trace matching in CLF. Technical Report CMU-CS-12-114, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, July 2012.

[13] Jeffrey Sarnat. *Syntactic Finitism in the Metatheory of Programming Languages*. PhD thesis, Yale University, May 2010.

[14] Anders Schack-Nielsen. *Implementing Substructural Logical Frameworks*. PhD thesis, IT University of Copenhagen, January 2011.

[15] Anders Schack-Nielsen and Carsten Schürmann. Celf — A logical framework for deductive and concurrent systems (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 320–326. PUB-SP, 2008.

[16] Anders Schack-Nielsen and Carsten Schürmann. Pattern unification for the lambda calculus with linear and affine types. In Karl Crary and Marino Miculan, editors, *LFMTP*, volume 34 of *EPTCS*, pages 101–116, 2010.

[17] Robert J. Simmons. *Substructural Logical Specifications*. PhD thesis, Carnegie Mellon University, 2012.

[18] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A Concurrent Logical Framework I: Judgments and Properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 2003.