Ill-Typed Programs Don't Evaluate

STEVEN RAMSAY, University of Bristol, UK
CHARLIE WALPOLE, University of Bristol, UK

We introduce two-sided type systems, which are a particular kind of sequent calculi for typing formulas. Two-sided type systems allow for hypothetical reasoning over the typing of compound program expressions, and the refutation of typing formulas. By incorporating a type of all values, these type systems support symmetrical notions of well-typing and ill-typing, guaranteeing both that well-typed programs don't go wrong and that ill-typed programs do not evaluate - that is, reach a value. This makes two-sided type systems suitable for incorrectness reasoning in higher-order program verification, which we illustrate through an application to precise data-flow typing in a language with constructors and pattern matching. Finally, we investigate the internalisation of the meta-level negation in the system as a complement operator on types. This motivates an alternative semantics for the typing judgement, which guarantees that ill-typed programs don't evaluate, but in which well-typed programs may yet go wrong.

Additional Key Words and Phrases: type systems, higher-order program verification, incorrectness

1 INTRODUCTION

It is natural to think of a type system as a kind of proof system, whose purpose is reasoning about the behaviour of program expressions. In this view, the atomic formulas of the system are typings $M : A$, where M is a term and A a type and, in building a proof of a judgement $\Gamma \vdash M : A$, we aim to conclude an atomic formula $M : A$ under some assumptions Γ on the types of its free variables. Therefore, built into traditional type systems, but absent from most general proof systems, is a fundamental asymmetry: although we can conclude arbitrary typing formulas $M : A$, we may only make assumptions on variable typing formulas $x : B$. In this paper, we argue that type theory is enriched by removing this restriction, and there are interesting practical applications. Our *two-sided type systems* allow for making assumptions on the types of arbitrary program expressions. In fact, they are particular kinds of sequent calculi over formulas of shape $M : A$.

The ability to reason hypothetically about the behaviour of compound terms gives us a new way to express how programs depend on their inputs. For example, we can prove that, if $x + 1$ evaluates to a natural number and $(\lambda f z.\, f\ (f\ z))\ (\lambda x.\, x)\ y$ evaluates to a natural number, then (x, y) is necessarily a pair of natural numbers:

$$x + 1 : \mathsf{Nat},\ (\lambda f z.\, f\ (f\ z))\ (\lambda x.\, x)\ y : \mathsf{Nat} \vdash (x, y) : \mathsf{Nat} \times \mathsf{Nat}$$

We also remove the restriction that one must conclude exactly one typing formula on the right of the turnstile, with multiple formulas understood disjunctively (as in most sequent calculi). In particular, one is allowed to derive an empty conclusion $\Gamma \vdash$, meaning that the assumed typing formulas Γ are inconsistent (the unit of disjunction is absurdity). For example, one can prove:

$$(\lambda f z.\, f\ (f\ z))\ (\lambda x.\, x)\ (y,\ y) : \mathsf{Nat} \vdash$$

stating that, no matter the value of y, the subject on the left *doesn't* evaluate to a natural number.

Despite this, the term above *does* evaluate, i.e. reduce to a value. By introducing a type Ok to characterise all values, we can use hypothetical judgements like the above to *prove* that terms *do not evaluate* (at all), that is they either diverge or go wrong. For example, we can prove:

$$(\lambda f z.\, f\ (f\ z))\ (\lambda x.\, \mathsf{pred}(x))\ (y,\ y) : \mathsf{Ok} \vdash$$

stating that the program that applies the predecessor function twice in succession to a pair will fail to evaluate. We say that such a term, to which we can assign the type Ok on the left, is *ill-typed*.

As well as the familiar rules for concluding typings on the right of the turnstile, our system also has rules for refuting assumed typings on the left. The key to the whole enterprise is a new kind of function type, the necessity arrow $A \succ\!\!- B$, pronounced '*A only to B*'. The necessity arrow describes functions that produce a B *only if* they are given an A as input. It is, in a sense that we make precise, the contrapositive of the usual (sufficiency) arrow $A \to B$, which guarantees that a B is produced *if* an A is given as input. The rule, (AbnR), for deducing that a function $\lambda x.\, M$ is of type $A \succ\!\!- B$, is symmetrical to the usual rule for $A \to B$:

$$\text{(AppL)}\ \frac{\Gamma \vdash M : A \succ\!\!- B,\ \Delta \quad \Gamma,\ N : A \vdash \Delta}{\Gamma,\ (M\,N) : B \vdash \Delta} \qquad \text{(AbnR)}\ \frac{\Gamma,\ M : B \vdash x : A,\ \Delta}{\Gamma \vdash (\lambda x.\, M) : A \succ\!\!- B,\ \Delta}$$

On the other hand, the rule (AppL) shows us how to decompose an application on the left, using necessity. As is typical in sequent calculi, it is instructive to think of a left rule in terms of refuting its principal formula. To *refute* that an application $M\,N$ evaluates to an B, it suffices to *affirm* that M is a function that produces a B only if given an A, and then to *refute* that N is an A.

The ability to refute that terms evaluate becomes very useful when we consider more sophisticated type systems, designed for verifying stronger, behavioural properties of terms. In such systems, one faces the same difficulties as in general program verification, namely that proofs are difficult to construct automatically and, as a consequence, false positives – that is, where a perfectly good program is not well-typed[1] – are common. This is problematic because, in practice, a lot of the value of program verification tools derives from their use as a means to find bugs, i.e. true positives.

One might argue that false positives are already a feature of the mainstream type systems, as e.g. used in compilers, and are thus evidently not too onerous (though proponents of untyped programming may forcefully disagree). However, a significant difference is that, by and large, programmers understand enough of those systems to be able to predict false positives, or at least quickly diagnose and program around them. By contrast, type systems for program verification are often significantly more complex and the success of automated tools is bound up with the efficacy of *ad hoc* heuristics.

With two-sided typing, we can obtain a 'true positives' theorem: *ill-typed programs do not evaluate* in addition to the usual 'true negatives' theorem: *well-typed programs do not go wrong*. Thus, like O'Hearn's Incorrectness Logic for imperative programs [O'Hearn 2019], our two-sided typing provides some foundation for the use of type systems to predict erroneous program behaviours. Indeed, one of the original motivations for this work was to understand the basis for Erlang's highly effective Success Typing [Lindahl and Sagonas 2006].

We illustrate this in a two-sided, constrained type system designed for reasoning precisely about the shape of data structures. Following Aiken et al. [1994], our system has a type constructor for every datatype constructor, so that, for example, [] is both the term constructor for the empty list and the type of the empty list, and $A :: B$ is the type of all non-empty (cons ::) lists whose head element has type A and whose tail is of type B. Using the system we can prove, for example, that the head function requires a cons with head element of type A in order to return an A, and that the map function requires non-emptiness of its list argument in order to provide a non-empty output:

$$\mathsf{head} : \forall a.\, (a :: \mathsf{Ok}) \succ\!\!- a \qquad \mathsf{map} : \forall a\, b.\, (a \succ\!\!- b) \to (a :: \mathsf{Ok}) \succ\!\!- (b :: \mathsf{Ok})$$

More precisely, the combination of *to* and *only to* arrows in map's type says that when given an a only to b function, it is guaranteed to behave like a function that necessarily requires a cons with element type a in order to produce a cons with element type b. Since our system can refute that the

[1] A simple example that is rejected by many systems is: if true then 1 else (1, 1).

empty list is non-empty, taking the head of a list obtained by mapping the empty list is *ill-typed*:

$$\mathsf{head}\,(\mathsf{map}\,(\lambda x.\,x)\,[\,])\,:\,\mathsf{Ok} \vdash$$

This program, which typically shows as a *false negative* in mainstream type systems for functional programming (i.e. is reported as well-typed but actually goes wrong), is proven by our system to be a true positive: the term is guaranteed to either diverge or go wrong.

Our full list of contributions is as follows.

- We introduce a two-sided type system for a PCF-like language and we prove *semantic soundness*, i.e. that every provable judgement is true in the call-by-value semantics (Theorem 4.3). Soundness implies that *well-typed programs don't go wrong* and also that *ill-typed programs don't evaluate*. A crucial step is to show that all our types describe safety properties (Theorem 4.5). So, in contrast to e.g. incorrectness logic, we can use standard rules for reasoning about recursion via invariants. However, we show that this is lost when allowing higher-types on the right of the necessity arrow (Theorem 4.8).
- We introduce a two-sided constrained type system for a functional programming language with datatype constructors and pattern matching. The system allows for precise reasoning about the shape of data structures through expressive type inclusion constraints. We show that types can be inferred automatically for this system and are principal in a suitable sense (Theorem 5.8). Using an adaptation of the technique of Wright and Felleisen [1994], we prove *syntactic soundness*. Although weaker than semantic soundness, we obtain both that well-typed programs don't go wrong (as usual) and ill-typed programs don't evaluate (Theorem 5.12).
- Finally, we introduce an alternative *success semantics* for two-sided judgements, in which showing that a term is typable does not preclude that the term goes wrong. We show that, under the success semantics, the meta-level negation provided by the two-sided judgement can be internalised in the system as a complement operator on types. As a consequence, we can derive a one-sided (traditional) type system, which actually subsumes its two-sided cousin (Theorem 6.4). We prove syntactic soundness for the one-sided system, which implies that both systems guarantee that ill-typed programs don't evaluate, though well-typed programs may yet go wrong (Theorem 6.6).

The paper continues by introducing the basic ideas of two-sided typing through a simple, PCF-like programming language in Sections 2 and 3, before considering the semantics of such systems in Section 4. We then apply the ideas to precise reasoning in a functional programming language with constructors and pattern matching in Section 5, which we show is sound and has computable inference. Section 6 returns to the simple PCF-like language for an investigation into complements and the success semantics. Finally, Section 7 concludes with a review of related work.

2 A LANGUAGE AND ITS TWO-SIDED TYPING

We start by introducing a PCF-like, call-by-value programming language.

Definition 2.1 (Terms). Assume a denumerable set of term variables x, y, z and so on. We consider a simple functional programming language, whose *values*, typically V, W and so on, and whose *terms*, typically M, N, P, Q and so on, are defined by the following grammar:

$$\begin{array}{rcl} V,\,W & ::= & x \mid \mathsf{succ}^n(\mathtt{zero}) \mid (V,\,W) \mid \lambda x.\,M \\ M,N,P,Q & ::= & \mathtt{zero} \mid \mathsf{succ}(M) \mid \mathsf{pred}(M) \mid \mathsf{if}\,M\,\mathsf{then}\,N\,\mathsf{else}\,P \mid \\ & & \quad x \mid \lambda x.\,M \mid M\,N \mid \mathsf{fix}\,x.\,M \mid (M,N) \mid \mathsf{let}\,(x,y) = M\,\mathsf{in}\,N \end{array}$$

We consider terms identified up to renaming of bound variables and adopt the usual conventions regarding their treatment. A term with no free variables is said to be *closed*. A *term substitution*,

typically θ, is finite map from term variables to terms. We say that a substitution is *closed* just if the terms in its range are closed. We will often write concrete substitutions explicitly as a list of maplets as in $[M_1/x_1, \ldots, M_n/x_n]$, or $[\overrightarrow{M_i/x_i}]$ when indices are clear.

Abbreviations. It will be helpful to define some abbreviations. For each $n \in \mathbb{N}$, we write $\underline{n}$ for the *numeral* $\mathsf{succ}^n(\mathtt{zero})$. The let construct is used to obtain the two components of a pair by matching but, in examples, we will typically always use it on the argument of an abstraction. Hence, it is helpful to define $\lambda(x, y).M$ as an abbreviation for $\lambda z.\,\mathsf{let}\,(x, y) = z\,\mathsf{in}\,M$. We write $\underline{\mathsf{div}}$ as an abbreviation for $\mathsf{fix}\,x.\,x$, and $\underline{\mathsf{id}}$ for $\lambda x.\,x$.

The language is essentially an applied λ-calculus with fixpoints. Terms M can be tested for equality with $\mathtt{zero}$ using the if-zero expression $\mathsf{if}\,M\,\mathsf{then}\,N\,\mathsf{else}\,P$, and can be deconstructed as a pair using the let expression $\mathsf{let}\,(x, y) = M\,\mathsf{in}\,N$. Closed values are numerals, pairs and abstractions.

Definition 2.2 (Reduction). The *evaluation contexts* are defined by the following grammar:

$$\begin{aligned}\mathcal{E}, \mathcal{F} \quad ::= \quad & \Box \mid \mathsf{succ}(\mathcal{E}) \mid \mathsf{pred}(\mathcal{E}) \mid (\mathcal{E}, M) \mid (V, \mathcal{E}) \\ & \mid \mathsf{let}\,(x, y) = \mathcal{E}\,\mathsf{in}\,N \mid \mathsf{if}\,\mathcal{E}\,\mathsf{then}\,N\,\mathsf{else}\,P \mid (\lambda x.\,M)\,\mathcal{E} \mid \mathcal{E}\,N\end{aligned}$$

Given a context $\mathcal{E}$, we write $\mathcal{E}[M]$ for the term obtained by replacing the hole $\Box$ by M. The *one-step reduction relation*, written $M \rhd N$, is the binary relation on (possibly open) terms obtained as the closure of the following schema under evaluation contexts.

(IfZ)	$\mathsf{if}\,\underline{0}\,\mathsf{then}\,N\,\mathsf{else}\,P \rhd N$	(IfS)	$\mathsf{if}\,\underline{n+1}\,\mathsf{then}\,N\,\mathsf{else}\,P \rhd P$
(Let)	$\mathsf{let}\,(x, y) = (V, W)\,\mathsf{in}\,M \rhd M[V/x, W/y]$	(Fix)	$\mathsf{fix}\,x.\,M \rhd M[\mathsf{fix}\,x.\,M/x]$
(PZ)	$\mathsf{pred}(\underline{0}) \rhd \underline{0}$	(PS)	$\mathsf{pred}(\underline{n+1}) \rhd \underline{n}$
(Beta)	$(\lambda x.\,M)\,V \rhd M[V/x]$		

The terms on the left-hand side of the above schema are said to be *redexes*. We write $M \rhd^* N$ for the reflexive transitive closure of the one-step relation. A term M that cannot make a step is said to be in *normal form*. A term with no normal form is said to *diverge*. We write $M \Downarrow V$ just if closed term $M \rhd^* V$ and we say that M *evaluates*. A term is said to be *stuck* just if it is a normal form that is not a value. A term is said to *get stuck* (or *go wrong*) just if it reduces to a stuck term.

2.1 Two-Sided Type System

We introduce a very simple two-sided type system for this language. The types of the system are either the base type Nat, or constructed using product or the two arrows. We make a distinction between *finitely verifiable* types and ordinary types. Intuitively, membership in the set of values described by finitely verifiable types can be witnessed by a finite unfolding of (the recursion in) the term. We make this distinction to ensure that types describe safety properties (see Section 4).

Definition 2.3 (Types). The *types*, typically A, B and so on, and the *finitely verifiable types*, typically F, are defined by the following grammar:

$$\begin{aligned} F \quad &::= \quad \mathsf{Nat} \mid F_1 \times F_2 \\ A, B \quad &::= \quad \mathsf{Nat} \mid A \times B \mid A \to B \mid A \rightarrowtail F \end{aligned}$$

We will label the usual arrow type $A \to B$, the *sufficiency arrow* to distinguish it from the *necessity arrow* $A \rightarrowtail F$. We pronounce the first of these types as 'A to B' and the second as 'A only to B'[2]. We assume all arrows associate to the right and that products bind tighter than the other operators.

[2]Strictly speaking, for the English to be correct, one should pronounce it as 'A only, to B', but we do not recommend it.

Structural

$$\text{(Id)}\ \frac{}{\Gamma,\ x : A \vdash x : A,\ \Delta} \qquad \text{(Dis)}\ \frac{\Gamma \vdash M : B,\ \Delta}{\Gamma,\ M : A \vdash \Delta}\ A \mid\mid B$$

Right Rules

$$\text{(ZeroR)}\ \frac{}{\Gamma \vdash \mathsf{zero} : \mathsf{Nat},\ \Delta} \qquad \text{(SuccR)}\ \frac{\Gamma \vdash M : \mathsf{Nat},\ \Delta}{\Gamma \vdash \mathsf{succ}(M) : \mathsf{Nat},\ \Delta} \qquad \text{(PredR)}\ \frac{\Gamma \vdash M : \mathsf{Nat},\ \Delta}{\Gamma \vdash \mathsf{pred}(M) : \mathsf{Nat},\ \Delta}$$

$$\text{(LetR)}\ \frac{\Gamma \vdash M : B \times C,\ \Delta \qquad \Gamma,\ x : B,\ y : C \vdash N : A,\ \Delta}{\Gamma \vdash \mathsf{let}\ (x, y) = M\ \mathsf{in}\ N : A,\ \Delta}$$

$$\text{(AppR)}\ \frac{\Gamma \vdash M : B \to A,\ \Delta \qquad \Gamma \vdash N : B,\ \Delta}{\Gamma \vdash M\,N : A,\ \Delta} \qquad \text{(PairR)}\ \frac{\Gamma \vdash M : A,\ \Delta \qquad \Gamma \vdash N : B,\ \Delta}{\Gamma \vdash (M,\ N) : A \times B,\ \Delta}$$

$$\text{(AbsR)}\ \frac{\Gamma,\ x : B \vdash M : A,\ \Delta}{\Gamma \vdash \lambda x.\, M : B \to A,\ \Delta} \qquad \text{(AbnR)}\ \frac{\Gamma,\ M : F \vdash x : B,\ \Delta}{\Gamma \vdash \lambda x.\, M : B \rightarrowtail F,\ \Delta} \qquad \text{(FixR)}\ \frac{\Gamma,\ x : A \vdash M : A,\ \Delta}{\Gamma \vdash \mathsf{fix}\, x.\, M : A,\ \Delta}$$

$$\text{(IfZR)}\ \frac{\Gamma \vdash M : \mathsf{Nat},\ \Delta \qquad \Gamma \vdash N : A,\ \Delta \qquad \Gamma \vdash P : A,\ \Delta}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P : A,\ \Delta}$$

Left Rules

$$\text{(SuccL)}\ \frac{\Gamma,\ M : \mathsf{Nat} \vdash \Delta}{\Gamma,\ \mathsf{succ}(M) : \mathsf{Nat} \vdash \Delta} \qquad \text{(PredL)}\ \frac{\Gamma,\ M : \mathsf{Nat} \vdash \Delta}{\Gamma,\ \mathsf{pred}(M) : \mathsf{Nat} \vdash \Delta}$$

$$\text{(AppL)}\ \frac{\Gamma \vdash M : B \rightarrowtail A,\ \Delta \qquad \Gamma,\ N : B \vdash \Delta}{\Gamma,\ M\,N : A \vdash \Delta} \qquad \text{(PairL)}\ \frac{\Gamma,\ M_i : A_i \vdash \Delta}{\Gamma,\ (M_1,\ M_2) : A_1 \times A_2 \vdash \Delta}$$

$$\text{(LetL1)}\ \frac{\Gamma,\ N : A \vdash \Delta}{\Gamma,\ \mathsf{let}\ (x, y) = M\ \mathsf{in}\ N : A \vdash \Delta} \qquad \text{(LetL2)}\ \frac{\begin{array}{c}\Gamma,\ M : B_1 \times B_2 \vdash \Delta \\ \Gamma,\ N : A \vdash x_i : B_i,\ \Delta\ (\forall i)\end{array}}{\Gamma,\ \mathsf{let}\ (x_1, x_2) = M\ \mathsf{in}\ N : A \vdash \Delta}$$

$$\text{(IfZL1)}\ \frac{\Gamma,\ M : \mathsf{Nat} \vdash \Delta}{\Gamma,\ \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P : A \vdash \Delta} \qquad \text{(IfZL2)}\ \frac{\Gamma,\ N : A \vdash \Delta \qquad \Gamma,\ P : A \vdash \Delta}{\Gamma,\ \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P : A \vdash \Delta}$$

Fig. 1. Two-sided type assignment.

The idea of $A \rightarrowtail B$, which will be made precise in Section 4, is to classify those functions which return a B *only if* an A was supplied as input (hence the pronunciation). An example is the function $\lambda x.\,(x, \underline{0})$ which can be assigned the type $\mathsf{Nat} \rightarrowtail \mathsf{Nat} \times \mathsf{Nat}$. If this function returns a pair of numerals, its input must have been a numeral. On the other hand, $\lambda x.\,(\underline{0}, \underline{1})$ cannot be assigned this type, because if it returns a pair of numerals, it is *not necessary* that a numeral was input.

We shall not consider subtyping for this system, but it is useful to axiomatise a notion of type *disjointness* which, intuitively, says that two types do not have any values in common.

Definition 2.4 (Disjointness). We say that two types A and B are *disjoint*, and write $A \mid\mid B$, just if either (i) one is Nat and the other is not, or (ii) one is an arrow and the other is not, or (iii) one is a product and the other is not.

Now, we come to define the two-sided type system. The idea is that the system is a simple kind of sequent calculus whose only formulas are typings $M : A$.

Definition 2.5 (Type Assignment). A *typing formula*, or just *typing*, is a pair $M : A$ of a term M and type A. The term M is said to be the *subject* of the formula. A *typing judgement* is a pair of finite sets of typings, written $\Gamma \vdash \Delta$. A judgement is said to be *provable* (or *derivable*) according to the rules in Figure 1. We make some additional requirements on the occurrences of free variables that are omitted from the rules for typesetting reasons: in every use of (AbsR), (AbnR), (LetR), (LetL1), (LetL2) and (FixR), we require that the bound variables displayed do not occur freely in Γ or Δ.

In a traditional, one-sided type system, a judgement like $x : A,\ y : B \vdash P : C$ is conventionally understood as 'if x has type A and y has type B, then P has type C'. Here, we generalise this in two ways. First, we allow arbitrary terms on the left and not only variables. However, the judgement can be read in the same way. So $M : A,\ N : B \vdash P : C$ should be understood as:

'if M has type A and N has type B, then P has type C'.

Second, we allow for multiple typings (including none) on the right of the turnstile. These are understood disjunctively, so $M : A \vdash P : C,\ Q : D$ should be read as:

'if M has type A, then *either* P has type C *or* Q has type D'.

In particular, $M : A,\ N : B \vdash$, which has an empty conclusion, should be understood as 'if M has type A and N has type B then false', i.e. either M cannot have type A or N cannot have type B.

However, we must take care to say something about the meaning of 'M has type A' within a judgement. Since our language is call-by-value, there is an asymmetry between the left- and right-hand side. On the left-hand side of the turnstile 'M has type A' means M *evaluates to an* A, but on the right-hand side, it means *either* M *diverges or* M *evaluates to an* A. This is made precise in Section 4, but it is useful to have some intuitions already.

For example, suppose we have some term $\underline{\mathsf{add}}$ and we know that $\underline{\mathsf{add}} : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$. Then, we should expect that the following judgement is provable:

$$\mathsf{succ}(\mathsf{succ}(z)) : \mathsf{Nat},\ \ \mathsf{if}\ y\ \mathsf{then}\ \underline{0}\ \mathsf{else}\ \underline{1} : \mathsf{Nat} \vdash \underline{\mathsf{add}}\,(y, z) : \mathsf{Nat}$$

Intuitively, if $\mathsf{succ}(\mathsf{succ}(z))$ is a numeral, it must be that z was a numeral, and if the term $\mathsf{if}\ y\ \mathsf{then}\ \underline{0}\ \mathsf{else}\ \underline{1}$ evaluates to a numeral, it must be that y is a numeral too (recall that the conditional of the language requires the guard to be a numeral). Therefore, $\underline{\mathsf{add}}\,(y, z)$ should evaluate to a numeral. This is the kind of reasoning that we are trying to capture with the rules of Figure 1.

The rules have been divided into three kinds, *structural rules*, *right rules* and *left rules*. Typically, the rules are used bottom-to-top in order to construct a derivation. The structural rules express some features particular to the meaning of the two sides. The right rules should be familiar from simple type systems, and are for concluding typings on the right of the turnstile; they only differ in

that they allow for multiple conclusions. One can think of the purpose of left rules as for refuting typings on the left of the turnstile. However, in practice, they are more often used as a means to extract information from assumptions.

Structural Rules. The rule (Id) should be familiar. The only difference is that now we allow for multiple formulas in the conclusion, but since they are understood disjunctively, all is well. The (Dis) rule expresses a key structural property of the two-sided type system, which is that refuting that a term M evaluates to an A can be achieved by affirming that M either diverges or evaluates to a value of some disjoint type B.

Left Rules. The rule (SuccL) allows for refuting that a term of shape $\mathsf{succ}(M)$ has type Nat. The only way that $\mathsf{succ}(M)$ may *not* evaluate to a numeral is if M does not evaluate to a numeral, so refuting $\mathsf{succ}(M) : \mathsf{Nat}$ reduces to refuting $M : \mathsf{Nat}$. The rule (PredL) is similar.

The rule (AppL) says that, to refute that $M\,N$ evaluates to an A, we can do the following. First show that M either diverges or requires an input of type B in order to produce an A, then refute that N evaluates to a B. Notice the combination of affirmation and refutation required in this rule.

The rule (PairL) says that to refute that (M_1, M_2) evaluates to a value of type $A_1 \times A_2$ we need only refute that *one* component of the pair evaluates accordingly.

The rules (LetL1) and (LetL2) allow for reasoning about let expressions on the left. To refute that $\mathsf{let}\,(x, y) = M\,\mathsf{in}\,N$ evaluates to an A, one can refute that the body evaluates to an A (independently of the values of x and y); this is (LetL1). Alternatively, one can refute that M evaluates to a pair (V, W) of the appropriate kind to allow $N[V/x, W/y]$ to evaluate to an A. We use the 'if ... then ...' reading of the judgement, in order to explain how the rule implements this strategy. The first two premises of (LetL2) say that, if N evaluates to an A then it must be that x was a B and that y was a C. In other words, in showing these two premises, we are showing that for N to evaluate to an A as in the conclusion, (x, y) is necessarily a value of $B \times C$. Then in the third premise we refute that M evaluates to a pair of this form.

Finally, the rules (IfZL1) and (IfZL2) are used for reasoning about the if-zero expression on the left. To refute that $\mathsf{if}\,M\,\mathsf{then}\,N\,\mathsf{else}\,P$ evaluates to an A, we can refute that M evaluates to a numeral, as in (IfZL1). Alternatively, we can show that neither branch evaluates to an A, which is (IfZR2).

Notice that there are no rules for reasoning explicitly about abstractions or fixed points on the left. In the case of abstractions, to refute $\lambda x.\, M : B \to A$, we would want to refute something about the body M. However, it would be unsound to simply refute that the body M evaluates to a B, because the meaning of the $B \to A$ allows for this possibility – M may diverge and yet $\lambda x.\, M$ is a value of $B \to A$. Note, however, that it is possible to refute $\lambda x.\, M : A$ by using the disjointness rule (Dis). We could have included left rules for fixpoint expressions, but since they are typically used in conjunction with abstractions it is of limited use.

Pattern Abstraction. Recall that we treat $\lambda(x, y).\, M$ as an abbreviation for $\lambda z.\, \mathsf{let}\,(x, y) = z\,\mathsf{in}\,M$. In the following it will be efficient to use the derived typing rule:

$$(\text{PAbsNL})\ \frac{\Gamma,\, M : A \vdash x : B,\, \Delta \qquad \Gamma,\, M : A \vdash y : C,\, \Delta}{\Gamma \vdash \lambda(x, y).\, M : B \times C \rightarrowtail A,\, \Delta}$$

Example 2.6. We can define addition $\underline{\mathsf{add}} = \mathsf{fix}\, f.\, (\lambda(y, z).\, \mathsf{if}\, y\, \mathsf{then}\, z\, \mathsf{else}\, \mathsf{succ}\, (f\, (\mathsf{pred}(y), z)))$. To return a numeral, both its arguments must be numerals. First observe that, in the 'else' branch of the conditional, if we know that the recursive call requires a pair of numerals in order to produce

a numeral, then z must be a numeral, we use Γ to abbreviate $\{ f : \mathsf{Nat} \times \mathsf{Nat} \rightarrowtail \mathsf{Nat} \}$ in:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash f : \mathsf{Nat} \times \mathsf{Nat} \rightarrowtail \mathsf{Nat},\ z : \mathsf{Nat}}\,(\mathrm{Id}) \qquad \dfrac{\dfrac{}{\Gamma,\ z : \mathsf{Nat} \vdash z : \mathsf{Nat}}\,(\mathrm{Id})}{\Gamma,\ (\mathsf{pred}(y), z) : \mathsf{Nat} \times \mathsf{Nat} \vdash z : \mathsf{Nat}}\,(\mathrm{PairL})}{\Gamma,\ f\ (\mathsf{pred}(y), z) : \mathsf{Nat} \vdash z : \mathsf{Nat}}\,(\mathrm{AppL})}{\Gamma,\ \mathsf{succ}(f\ (\mathsf{pred}(y), z)) : \mathsf{Nat} \vdash z : \mathsf{Nat}}\,(\mathrm{SuccL})$$

The same can be said in the 'then' branch of the conditional, since the 'then' branch actually returns z. So we can use (IfZL2) to conclude that z is certainly a numeral whenever the conditional expression evaluates to a numeral:

$$\dfrac{\dfrac{}{\Gamma,\ z : \mathsf{Nat} \vdash z : \mathsf{Nat}}\,(\mathrm{Id}) \qquad \dfrac{\cdots}{\Gamma,\ \mathsf{succ}(f\ (\mathsf{pred}(y), z)) : \mathsf{Nat} \vdash z : \mathsf{Nat}}}{\Gamma,\ \mathsf{if}\ y\ \mathsf{then}\ z\ \mathsf{else}\ \mathsf{succ}(f\ (\mathsf{pred}(y), z)) : \mathsf{Nat} \vdash z : \mathsf{Nat}}\,(\mathrm{IfZL2})$$

Abbreviating the 'else' branch for brevity, we can obtain $y : \mathsf{Nat}$ from the fact that the conditional branches on y, using (IfZ1), and conclude using our derived rule (PAbnR) and the standard rule for fixpoints on the right:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma,\ y : \mathsf{Nat} \vdash y : \mathsf{Nat}}\,(\mathrm{Id})}{\Gamma,\ \mathsf{if}\ y\ \mathsf{then}\ z\ \mathsf{else}\ \ldots : \mathsf{Nat} \vdash y : \mathsf{Nat}}\,(\mathrm{IfZL1}) \qquad \dfrac{\cdots}{\Gamma,\ \mathsf{if}\ y\ \mathsf{then}\ z\ \mathsf{else}\ \ldots : \mathsf{Nat} \vdash z : \mathsf{Nat}}}{\Gamma \vdash \lambda(y, z).\ \mathsf{if}\ y\ \mathsf{then}\ z\ \mathsf{else}\ \mathsf{succ}(f\ (\mathsf{pred}(y), z)) : \mathsf{Nat} \times \mathsf{Nat} \rightarrowtail \mathsf{Nat}}\,(\mathrm{PAbnR})}{\vdash \underline{\mathsf{add}} : \mathsf{Nat} \times \mathsf{Nat} \rightarrowtail \mathsf{Nat}}\,(\mathrm{FixR})$$

Example 2.7. We can define the higher-order combinator $\underline{\mathsf{twice}} = \lambda f x.\ f\ (f\ x)$. It has the following property. For any type A, it guarantees to map all functions of type $A \rightarrowtail A$ to an output function of the same type. We abbreviate $\Gamma = \{f : A \rightarrowtail A\}$ in the proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash f : A \rightarrowtail A,\ x : A}\,(\mathrm{Id}) \qquad \dfrac{\dfrac{}{\Gamma \vdash f : A \rightarrowtail A,\ x : A}\,(\mathrm{Id}) \qquad \dfrac{}{\Gamma,\ x : A \vdash x : A}\,(\mathrm{Id})}{\Gamma,\ f\,x : A \vdash x : A}\,(\mathrm{AppL})}{\Gamma,\ f\ (f\,x) : A \vdash x : A}\,(\mathrm{Id})}{\Gamma \vdash \lambda x.\ f\ (f\,x) : A \rightarrowtail A}\,(\mathrm{AbnR})}{\vdash \lambda f x.\ f\ (f\,x) : (A \rightarrowtail A) \to A \rightarrowtail A}\,(\mathrm{AbsR})$$

Example 2.8. We can prove that applying predecessor twice to a pair does *not* evaluate to a natural number. From the foregoing example, we know that $\underline{\mathsf{twice}}\ (\lambda x.\ \mathsf{pred}(x))$ is guaranteed to behave like a function that, to produce a numeral, requires a numeral as input:

$$\dfrac{\dfrac{\cdots}{\vdash \underline{\mathsf{twice}} : (\mathsf{Nat} \rightarrowtail \mathsf{Nat}) \to \mathsf{Nat} \rightarrowtail \mathsf{Nat}} \qquad \dfrac{\dfrac{\dfrac{}{x : \mathsf{Nat} \vdash x : \mathsf{Nat}}\,(\mathrm{Id})}{\mathsf{pred}(x) : \mathsf{Nat} \vdash x : \mathsf{Nat}}\,(\mathrm{PredL})}{\vdash (\lambda x.\ \mathsf{pred}(x)) : \mathsf{Nat} \rightarrowtail \mathsf{Nat}}\,(\mathrm{AbnR})}{\vdash \underline{\mathsf{twice}}\ (\lambda x.\ \mathsf{pred}(x)) : \mathsf{Nat} \rightarrowtail \mathsf{Nat}}\,(\mathrm{AppR})$$

Note, since our predecessor is a constant of fixed arity, we need to wrap it in an abstraction to provide it as an input. However, a pair is not a numeral, so we can use disjointness:

$$\dfrac{\dfrac{\cdots}{\vdash \underline{\mathsf{twice}}\ (\lambda x.\ \mathsf{pred}(x)) : \mathsf{Nat} \rightarrowtail \mathsf{Nat}} \qquad \dfrac{\dfrac{\cdots}{\vdash (\underline{0},\ \underline{1}) : \mathsf{Nat} \times \mathsf{Nat}}}{(\underline{0},\ \underline{1}) : \mathsf{Nat} \vdash}\,(\mathrm{Dis})}{\underline{\mathsf{twice}}\ (\lambda x.\ \mathsf{pred}(x))\ (\underline{0},\ \underline{1}) : \mathsf{Nat} \vdash}\,(\mathrm{AppL})$$

$$\text{(OkVarR)}\ \frac{}{\Gamma \vdash x : \mathsf{Ok},\ \Delta} \qquad \text{(OkL)}\ \frac{\Gamma,\ M : \mathsf{Ok} \vdash \Delta}{\Gamma,\ M : A \vdash \Delta} \qquad \text{(OkR)}\ \frac{\Gamma \vdash M : A,\ \Delta}{\Gamma \vdash M : \mathsf{Ok},\ \Delta}$$

$$\text{(OkApL1)}\ \frac{\Gamma,\ M : \mathsf{Ok} \rightarrowtail A \vdash \Delta}{\Gamma,\ M\,N : \mathsf{Ok} \vdash \Delta} \qquad \text{(OkApL2)}\ \frac{\Gamma,\ N : \mathsf{Ok} \vdash \Delta}{\Gamma,\ M\,N : \mathsf{Ok} \vdash \Delta}$$

$$\text{(OkSL)}\ \frac{\Gamma,\ M : \mathsf{Nat} \vdash \Delta}{\Gamma,\ \mathsf{succ}(M) : \mathsf{Ok} \vdash \Delta} \qquad \text{(OkPL)}\ \frac{\Gamma,\ M : \mathsf{Nat} \vdash \Delta}{\Gamma,\ \mathsf{pred}(M) : \mathsf{Ok} \vdash \Delta} \qquad \text{(OkPrL)}\ \frac{\Gamma,\ M_i : \mathsf{Ok} \vdash \Delta}{\Gamma,\ (M_1,\ M_2) : \mathsf{Ok} \vdash \Delta}$$

Fig. 2. Type Assignment with Ok

3 THE TYPE OF VALUES AND ILL TYPEDNESS

In fact, not only does $\underline{\mathsf{twice}}\ (\lambda x.\,\mathsf{pred}(x))\ (\underline{0},\ \underline{1})$ not evaluate a numeral, it does not evaluate at all – it goes wrong. In this section we enrich our system with the means to express this by introducing a new type Ok, with meaning 'evaluates (to a value)'.

Definition 3.1. We extend the grammar of types with the constant Ok and extend the typing rules to include those given in Figure 2.

The rule (OkVarR) expresses the fact that, in call-by-value, we may assume that all variables denote values. The rules (OkL) and (OkR) express the fact that Ok is the type of all values. To conclude that $M : \mathsf{Ok}$, it suffices to show $M : A$ since every type A represents some non-empty set of values. To refute that M evaluates to an A, it suffices to refute that it evaluates at all. Rules (OkApL1) and (OkApL2) express the fact that, for an application $M\,N$ to evaluate, requires that M evaluates to a function (see Remark 3.1) and that N evaluates. Rules (OkSL) and (OkPL) express the fact that the successor and predecessor are only defined on numerals. In combination with (OkL), these rules subsume (SuccL) and (PredL) respectively. Finally, (OkPrL) allows to refute that (M_1, M_2) evaluates by refuting that one of the components evaluates.

Remark 3.1. Since Ok is meant to be the set of all values, the set of all abstractions can be represented by any type of shape $\mathsf{Ok} \rightarrowtail A$. Indeed, using (AbnR) and (OkVarR) we can prove $\vdash \lambda x.\,M : \mathsf{Ok} \rightarrowtail A$.

For example, if we use (OkPL) instead of (PredL) in the first derivation of Example 2.8 we can show that $\vdash \underline{\mathsf{twice}}\ (\lambda x.\,\mathsf{pred}(x)) : \mathsf{Nat} \rightarrowtail \mathsf{Ok}$. Then, using (AppL) and (Dis), we can conclude $\underline{\mathsf{twice}}\ (\lambda x.\,\mathsf{pred}(x))\ (\underline{0}, \underline{1}) : \mathsf{Ok} \vdash$.

In Example 2.6, we proved that $\underline{\mathsf{add}}$ requires both its arguments to be numerals in order to return a numeral. However, we cannot prove that it requires both its arguments to be numerals in order to evaluate. To see why, suppose the body of the function, if y then z else $\mathsf{succ}\,(f\ (\mathsf{pred}(y), z))$, evaluates for some actual parameters y and z. Clearly, y is required to be a numeral since it appears in the guard. However, z is *not* required to be a numeral by the branches of the conditional. In particular, the 'then' branch yields $z : \mathsf{Ok} \not\vdash z : \mathsf{Nat}$. However, there is a good reason, it is possible for an application of the function to successfully evaluate even when the second argument is not a numeral! For example, $\underline{\mathsf{add}}(\underline{0}, \lambda x.\,x) \Downarrow \lambda x.\,x$. The best we can do is prove $\vdash \underline{\mathsf{add}} : \mathsf{Nat} \times \mathsf{Ok} \rightarrowtail \mathsf{Ok}$.

Some authors use the phrase 'ill-typed' as synonymous with untypable, but we will use it to mean that a term is provably not ok (as is suggestive of the suffix *-typed*).

Definition 3.2. Suppose M is a closed term.

- We say that M is *well-typed* just if $\vdash M : \mathsf{Ok}$.
- We say that M is *ill-typed* just if $M : \mathsf{Ok} \vdash$.

Of course, due to the (OkR) rule, the definition of well-typed subsumes the usual one. Note that M being well-typed does not imply that M evaluates, only that M does not get stuck: having type A on the right of the turnstile only means that M either diverges or M evaluates to an A (see Section 4 for details). On the other hand, proving that a closed term M is ill-typed can be understood as proving that M *does not* evaluate: $M : \mathsf{Ok} \vdash$ means that M does not evaluate to a value in Ok, so M either diverges or goes wrong. The following example is ill-typed only because it will diverge:

$$\dfrac{\dfrac{\dfrac{}{x : \mathsf{Nat} \succ\!\!- \mathsf{Ok} \vdash x : \mathsf{Nat} \succ\!\!- \mathsf{Ok}}\,(\mathrm{Id})}{\vdash \mathsf{fix}\, x.\, x : \mathsf{Nat} \succ\!\!- \mathsf{Ok}}\,(\mathrm{FixR}) \qquad \dfrac{\dfrac{\dfrac{}{y : \mathsf{Nat} \vdash y : \mathsf{Nat}}\,(\mathrm{Id})}{\vdash \lambda y.\, y : \mathsf{Nat} \to \mathsf{Nat}}\,(\mathrm{AbsR})}{\lambda y.\, y : \mathsf{Nat} \vdash}\,(\mathrm{Dis})}{(\mathsf{fix}\, x.\, x)\ (\lambda y.\, y) : \mathsf{Ok} \vdash}\,(\mathrm{AppL})$$

4 SEMANTICS OF TYPE ASSIGNMENT

In Sections 2 and 3, we gave some intuitions about how we want to understand the *meaning* of types and the typing judgement. We now make this precise.

Definition 4.1 (Semantics of Types). Let Vals_0 be the set of all closed values. We interpret types as certain sets of closed, well-behaved terms:

$$\begin{array}{rcl}
[\![\mathsf{Ok}]\!] & = & \mathsf{Vals}_0 \\
[\![\mathsf{Nat}]\!] & = & \{\ \underline{0}, \underline{1}, \underline{2}, \dots\ \} \\
[\![A \times B]\!] & = & \{\ (V, W) \mid V \in [\![A]\!],\ W \in [\![B]\!]\ \} \\
[\![A \to B]\!] & = & \{\ \lambda x.\, M \mid \forall V \in \mathsf{Vals}_0.\, V \in [\![A]\!] \Rightarrow M[V/x] \in \mathcal{T}_\perp[\![B]\!]\ \} \\
[\![A \succ\!\!- B]\!] & = & \{\ \lambda x.\, M \mid \forall V \in \mathsf{Vals}_0.\, M[V/x] \in \mathcal{T}[\![B]\!] \Rightarrow V \in [\![A]\!]\ \} \\
\\
\mathcal{T}[\![A]\!] & = & \{\ M \mid M \Downarrow V,\ V \in [\![A]\!]\ \} \\
\mathcal{T}_\perp[\![A]\!] & = & \{\ M \mid M \in \mathcal{T}[\![A]\!] \vee M \Uparrow\ \}
\end{array}$$

The idea is that $[\![A]\!]$ is the set of all closed values of type A, $\mathcal{T}[\![A]\!]$ is the set of all terms that would evaluate to a value of type A and $\mathcal{T}_\perp[\![A]\!]$ is the set of all terms that either evaluate to a value of type A or diverge. It follows that $\mathcal{T}_\perp[\![\mathsf{Ok}]\!]$ is the set of closed terms that do not go wrong. As usual, the meaning of the sufficiency arrow $A \to B$ is as the set of all functions that guarantee to map each A value to a B value or diverge in the process. The meaning of the necessity arrow $A \succ\!\!- B$ is as the set of all functions that require a value from A in order to eventually return a value from B.

Remark 4.1. It seems there is an asymmetry between the semantics of the two arrows, but this is only because of the bias towards values induced by CBV. Note that the two implications are equivalent to: $V \in \mathcal{T}[\![A]\!] \Rightarrow M[V/x] \in \mathcal{T}_\perp[\![B]\!]$ and $M[V/x] \in \mathcal{T}[\![B]\!] \Rightarrow V \in \mathcal{T}_\perp[\![A]\!]$. The relationship between the two function types will become clearer in Section 6.

A consequence of the semantics of necessity is that it can be a little subtle to express properties of interest. Given an function $\lambda x.\, M$, its membership in some type $[\![A \succ\!\!- F]\!]$ becomes trivial whenever there is no value V such that $M[V/x] \in \mathcal{T}[\![F]\!]$. This can occur, for example, when M simply never produces an F. Thus we have: $\lambda x.\, \mathsf{pred}(x) \in [\![\mathsf{Nat} \times \mathsf{Nat} \succ\!\!- \mathsf{Nat} \times \mathsf{Nat}]\!]$, because $\lambda x.\, \mathsf{pred}(x)$ does not return pairs. This phenomenon can often be exploited in the proof system, using the disjointness rule. For example, using (Dis) we can prove $\vdash \lambda x.\, \mathsf{pred}(x) : \mathsf{Nat} \times \mathsf{Nat} \succ\!\!- \mathsf{Nat} \times \mathsf{Nat}$.

Now, we turn to the semantics of the typing judgement. Roughly speaking, we want that $M : A$ on the left means that M is a term that evaluates to an A, and $M : A$ on the right means that M is a term that either evaluates to an A or diverges.

Definition 4.2 (Semantics of Judgements). A *valuation*, θ, is just a closed substitution.

- Given an atomic formula $M : A$, we say that a valuation θ *satisfies the formula on the left* just if $M\theta \in \mathcal{T}[\![A]\!]$. We say that θ *satisfies the formula on the right* just if $M\theta \in \mathcal{T}_\perp[\![A]\!]$.
- We say that a valuation θ *satisfies a set of formulas* Γ *on the left*, just if θ satisfies each formula in Γ on the left. A valuation θ *satisfies a set of formulas* Δ *on the right*, just if there is a formula in Δ that is satisfied on the right.
- We say that a judgement $\Gamma \vdash \Delta$ is *true*, written $\Gamma \models \Delta$, just if all valuations θ that satisfy Γ on the left, satisfy Δ on the right.

To keep the development smooth, whenever we talk about the satisfaction of formulas by some valuation θ, we will assume that θ closes all the terms involved (i.e. its domain is sufficiently large).

Notice from the first of the three clauses that divergence is always allowed on the right, but never on the left. So, for example, valuation $[\underline{1}/x]$ satisfies $\mathsf{if}\, x\, \mathsf{then}\, \underline{0}\, \mathsf{else}\, \underline{\mathsf{div}} : \mathsf{Nat}$ on the right but not on the left. Then, notice also the asymmetry in the semantics of the judgement: $\Gamma \models \Delta$ just if all valuations that satisfy *all* formulas in Γ on the left, satisfy *some* formula from Δ on the right.

4.1 Soundness

We now look towards proving the soundness of this system with respect to this semantics, i.e. that any provable judgement $\Gamma \vdash \Delta$ is true. More precisely, we call this result *semantic soundness* to contrast with the weaker result obtained using a progress/preservation argument in Section 5.3.

Theorem 4.3 (Semantic Soundness). *If* $\Gamma \vdash \Delta$ *then* $\Gamma \models \Delta$.

The proof is by induction on the typing relation. Hence, it consists of showing, for each typing rule, that when the premises are true (and any side conditions hold) then the conclusion is also true. For almost all rules, this is straightforward.

However, the fixpoint typing rule presents certain difficulties to this syntactical semantics of typing. The problem is that to use the premise $\Gamma, x : A \models M : A, \Delta$, we need to invent a suitable value in $[\![A]\!]$. Intuitively, we would like to take this value in $[\![A]\!]$ as the fixed point of M (viewed as a function of x) but, on the face of it, there is no evidence that $\mathsf{fix}\, x.\, M$ is an A. Hence, we are forced to examine the structure of types and the fixpoint in a little more detail.

Definition 4.4. Given a fixpoint term $\mathsf{fix}\, x.\, M$, the following family of terms, indexed by $n \in \mathbb{N}$, are its *fixpoint approximants*.

$$\mathsf{fix}^0 x.\, M = \underline{\mathsf{div}}$$
$$\mathsf{fix}^{n+1} x.\, M = M[\mathsf{fix}^n x.\, M/x]$$

Note: we are not introducing a new piece of syntax, but merely an abbreviation that will be useful in the sequel. Each approximant is simply a finite unfolding of the possibly infinite fixpoint computation, with divergence (a complete lack of stable information) as the base case. The idea is to think of a term $C[\mathsf{fix}\, x.\, M]$ containing a fixpoint as being finitely approximated by each member of the family $C[\mathsf{fix}^n x.\, M]$. Next, we observe that types represent *safety properties*.

Theorem 4.5 (Types are safety properties.). *For closed terms* N *and* P*, write* $P \lesssim N$ *just if* P *normalises implies* N *normalises and then they have the same normal form.*

(i) If $C[N] \in \mathcal{T}_\perp[\![A]\!]$ *and* $P \lesssim N$*, then* $C[P] \in \mathcal{T}_\perp[\![A]\!]$.

(ii) If $C[\mathsf{fix}\, x.\, M] \notin \mathcal{T}_\perp[[A]]$*, then there is already some finite* k *for which* $C[\mathsf{fix}^k x.\, M] \notin \mathcal{T}_\perp[[A]]$.

The first clause states that typing (on the right) is closed under the introduction of divergence: $C[N]$ and $C[P]$ behave similarly, except $C[P]$ may diverge more often. This corresponds to the intuition that safety properties are about partial correctness. The second clause states that if a term involving a fixpoint does not satisfy a type, then one can already find a finite approximation of the term that fails to satisfy it. This corresponds to the intuition that when safety properties are violated, one can always find a finite counterexample.

The contrapositive of the second of these properties gives us a principle that we can use in order to show that term does satisfy a type: if $\mathsf{fix}^n x.\, M \in \mathcal{T}_\perp[[A]]$ for all n, then it follows that $\mathsf{fix}\, x.\, M \in \mathcal{T}_\perp[[A]]$. We use this principle to show the following.

Theorem 4.6 (Typing closed under fixpoints.). *If* $\lambda x.\, M \in [[A \to A]]$*, then* $\mathsf{fix}\, x.\, M \in \mathcal{T}_\perp[[A]]$.

With this result, we can complete the difficult case of Theorem 4.3, because the truth of the judgement $x : A \vdash M : A$ amounts to the fact that $\lambda x.\, M \in [[A \to A]]$. As corollary, we obtain:

Corollary 4.7. *Suppose* M *is a closed term.*

- *If* M *is well typed, then* M *does not go wrong.*
- *If* M *is ill typed, then* M *does not evaluate.*

4.2 Beyond safety

The proof of semantic soundness (Theorem 4.3) relies crucially on the fact that types are safety properties (Theorem 4.5). Here, we present two counterexamples to show that, if we were to allow higher-types on the right of the necessity arrow, then we would depart from the world of safety properties, and thus change the character of the type system considerably. The first is quite straightforward, but the second is a little tricky.

Theorem 4.8. *Suppose we extend the type language to allow all types of shape* $A \rightarrowtail B$.

- *The type* $\mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}$ *is not downwards closed.*
- *The type* $\mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}$ *does not furnish finite counterexamples.*

Proof. We prove each separately.

- The term $\lambda xy.\, (\lambda z.\, \mathsf{pred}(z))\, x \in \mathcal{T}_\perp[[\mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}]]$ because, reasoning according to the contrapositive of the definition, if a given value V is not a numeral, then it follows that $\lambda y.\, (\lambda z.\, \mathsf{pred}(z))\, V \notin \mathcal{T}[[\mathsf{Nat} \to \mathsf{Nat}]]$, since the body will get stuck on any argument. However, replacing $\lambda z.\, \mathsf{pred}(z)$ by $\underline{\mathsf{div}}$, we have that $\lambda xy.\, \underline{\mathsf{div}}\, x \notin \mathcal{T}_\perp[[\mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}]]$. Even when a given value V is not a numeral, we yet have $\lambda y.\, \underline{\mathsf{div}}\, V \in \mathcal{T}[[\mathsf{Nat} \to \mathsf{Nat}]]$, since the body diverges on any input.
- First observe that $\lambda wxyz.\, (\mathsf{fix}\, f.\, \lambda u.\, \mathsf{if}\, u\, \mathsf{then}\, \underline{\mathsf{add}}(y, z)\, \mathsf{else}\, f\, (\mathsf{pred}(u)))\, x$ is not a value of the type $\mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}$. To see why, let us abbreviate the conditional subterm by M and show that when applied to a value $\underline{\mathsf{id}}$ that is not a numeral, it may yet behave like a function of type $\mathsf{Nat} \to \mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}$. To see that $\lambda xyz.\, (\mathsf{fix}\, f.\, \lambda u.\, M)\, x$ is a value in $[[\mathsf{Nat} \to \mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}]]$, we let $n \in \mathbb{N}$ and show that $\lambda yz.\, (\mathsf{fix}\, f.\, \lambda u.\, M)\, \underline{n}$ is a value in $[[\mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}]]$. To see this, we use the contrapositive of the definition and show that when applied to any non-numeral value, the function returned does not satisfy $\mathsf{Nat} \to \mathsf{Nat}$. So let $V \notin [[\mathsf{Nat}]]$ be a non-numeral value. Then the function $\lambda z.\, (\mathsf{fix}\, f.\, \lambda u.\, \mathsf{if}\, u\, \mathsf{then}\, \underline{\mathsf{add}}(V, z)\, \mathsf{else}\, f\, (\mathsf{pred}(u)))\, \underline{n}$ is not in $[[\mathsf{Nat} \to \mathsf{Nat}]]$ because, when applied to $\underline{0}$ it will reduce to $\underline{\mathsf{add}}(V, \underline{0})$, but V is not a numeral.

  Next, observe that any term $\lambda wxyz.\, (\mathsf{fix}^k f.\, \lambda u.\, \mathsf{if}\, u\, \mathsf{then}\, \underline{\mathsf{add}}(y, z)\, \mathsf{else}\, f\, (\mathsf{pred}(u)))\, x$, in which

we have replaced the fixpoint by a particular finite approximant, *is* a value of the type $\mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}$. To see this, let us abbreviate the conditional subterm again by M and let $W \notin [[\mathsf{Nat}]]$. We show that, therefore, $\lambda xyz.\, (\mathsf{fix}^k f.\, \lambda u.\, M)\ x$ is not in $[[\mathsf{Nat} \to \mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}]]$. Take $x := \underline{k+1}$ as witness. Then $\lambda yz.\, (\mathsf{fix}^k f.\, \lambda u.\, M)\ \underline{k+1}$ is not in $[[\mathsf{Nat} \rightarrowtail \mathsf{Nat} \to \mathsf{Nat}]]$. To see why, take $y := \underline{\mathsf{id}} \notin [[\mathsf{Nat}]]$ as the witness, and we find that yet $\lambda z.\, (\mathsf{fix}^k f.\, \lambda u.\, \mathsf{if}\ u\ \mathsf{then}\ \underline{\mathsf{add}}(\underline{\mathsf{id}}, z)\ \mathsf{else}\ f\ (\mathsf{pred}(u)))\ \underline{k+1}$ is actually in the type $\mathsf{Nat} \to \mathsf{Nat}$, because when given any numeral as input, it will diverge.

□

5 A CONSTRAINED TYPE SYSTEM

In this section, we temporarily move away from our simple, PCF-like programming language and two-sided system to something more sophisticated. Our aim is to show: how necessity can be useful in practice, a syntactic soundness result, and how to infer types automatically.

Definition 5.1. We assume a denumerable set of *top-level identifiers*, written typically as f, g, and a denumerable set of *local variables*, typically f, g, x, y, z. We also fix a finite signature C of ranked datatype *constructors*, typically c. We will always assume that the binary pair constructor, written mixfix as $(\ ,\ )$ is in the signature. We consider various kinds of program expressions:

$$\begin{array}{rcl} p,\, q & ::= & c(x_1, \ldots, x_m) \\ M,\, N,\, P,\, Q & ::= & x \mid c(M_1, \ldots, M_m) \mid M\,N \mid \lambda x.\, M \mid \mathsf{fix}\, x.\, M \mid \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\} \\ V,\, W & ::= & x \mid \lambda x.\, M \mid c(V_1, \ldots, V_n) \\ \mathcal{M} & ::= & \epsilon \mid f = M;\ \mathcal{M} \end{array}$$

As usual, we identify terms up to renaming of bound variables. We consider a term to be *closed* just if it contains no free variables (but it may contain top-level function identifiers).

We make the following additional requirements: (i) in a pattern $c(x_1, \ldots, x_n)$ the arity of c is n and the x_i are pairwise distinct, (ii) in a pattern-match term $\mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\}$, the family $(p_i)_{i=1}^{k}$ is orthogonal (i.e. for every distinct i and j, p_i and p_j are headed by distinct constructors) and *all* bound variables throughout the family have distinct names, (iii) in module $f_1 = M_1; \ldots; f_n = M_n$, each f_i is distinct (there are not two definitions for the same top-level identifier).

Thus, we will feel free to index pattern families by the set of constructors, say I, that head their alternatives $|_{c \in I}\, c(\overrightarrow{x_c})$, and sometimes treat a module $\mathcal{M}$ as a partial map from identifiers to terms.

The first category are the *patterns*, which are required to be *simple* in the sense of being shallow and constructor headed. The second category are the *terms*. As in our PCF-like language, we have an applied λ-calculus with fixpoints, but this time we are interested in datatype constructors and pattern matching and thus we have constants accordingly. Thirdly, we have *modules* $\mathcal{M}$, which are just sequences of definitions for top-level identifiers. Finally, *values* are either variables, abstractions or datatype constructor terms built entirely from values.

Remark 5.1. The requirement for simple matching is to ease the presentation. We could allow arbitrary patterns at the expense of more complex subtype and consistency checking. Since this is not the focus of our work and is anyway well-covered elsewhere, we will be content with the simple case. Note also that the rank (arity) of a constructor is built into the syntax, so we will never consider the possibility of supplying a constructor with an inappropriate number of arguments since it is not considered a well-formed term.

Example 5.2. Let us assume a constructor signature C, which, in addition to the binary pair constructor $(\ ,\ )$, contains the binary list constructor 'cons', written infix ::, and the nullary empty

list constructor 'nil', written []. We can define the head and map functions by:

$$\begin{array}{rcl} \mathsf{head} & = & \lambda xs.\, \mathsf{match}\ xs\ \mathsf{with}\ \{y :: ys \mapsto y\} \\ \mathsf{map} & = & \mathsf{fix}\, m.\, \lambda f.\, \lambda xs.\, \mathsf{match}\ xs\ \mathsf{with}\ \{[] \mapsto [] \mid y :: ys \mapsto f\, y :: m\, f\, ys\} \end{array}$$

As before, we adopt a call-by-value reduction strategy and reuse the notation[3] $M \triangleright N$. The sequence of arguments to a datatype constructor is evaluated from left to right. Due to the requirement for shallow and orthogonal patterns, matching can be executed by simply indexing into the pattern family. A full definition is available in Appendix C.

5.1 Two-sided Constrained Type System

Our type system is designed for reasoning about the shape of datatype constructions. It is a *constrained type system* (see e.g. [Odersky et al. 1999]), so each judgement is parametrised by a set of type constraints C, which restrict the possible instantiations of type variables.

Our types are a cut-down version of the types defined in the constrained type system of Aiken et al. [1994], in that for each datatype constructor c of arity n, we can form a corresponding constructor type $c(A_1, \ldots, A_n)$. Intuitively, this type represents the set of all values of shape $c(V_1, \ldots, V_n)$, where each V_i has type A_i. We allow for universal polymorphism through the construction of *constrained type schemes*, $\forall\, \vec{a}\,.\, C \Rightarrow A$. Intuitively, a top-level function that has a such a scheme guarantees that to behaves like $A[\vec{B}/\vec{a}]$ for each instantiation $\vec{B}$ of the type variables $\vec{a}$ that satisfy the constraints C.

Definition 5.3. We assume a denumerable collection of *type variables* a, b and so on. The types are stratified as follows:

$$\begin{array}{rrcl} \text{(Sum Types)} & K & ::= & \Sigma_{i=1}^{k} c_i(\overrightarrow{A_i}) \\ \text{(Monotypes)} & A,\, B & ::= & a \mid K \mid \mathsf{Ok} \mid A \to B \mid A \rightarrowtail B \\ \text{(Type Schemes)} & S & ::= & A \mid \forall\, \vec{a}\,.\, C \Rightarrow A \end{array}$$

Here C (and sometimes D) is a finite set of *type constraints* (also *subtype formulas*), each of shape $A \sqsubseteq B$ (i.e. between monotypes). We write $A \equiv B$ as an abbreviation for the two constraints $A \sqsubseteq B$ and $B \sqsubseteq A$. We identify type schemes up to the renaming of bound type variables and we assume that arrows associate to the right. We require that type schemes are *closed* in the sense that they have no free type variables. Note that we *don't* restrict to finitely verifiable types on the right of necessity, which can be avoided for syntactic soundness – see the remark at the end of Section 5.3.

We consider the sum type $\Sigma_{i=1}^{k} c_i(\overrightarrow{A_i})$ as a finite set $\{c_1(\overrightarrow{A_1}), \ldots, c_k(\overrightarrow{A_k})\}$, and we require that the elements are orthogonal in the sense that $c_i = c_j$ implies $i = j$. See also Remark 5.1. In examples, We will often write a particular sum type $\Sigma_{i=1}^{3} c_i(\overrightarrow{A_i})$ as a list of summands $c_1(\overrightarrow{A_1}) + c_2(\overrightarrow{A_2}) + c_3(\overrightarrow{A_3})$. In particular, when the sum is a singleton $\Sigma_{i=1}^{1} c_i(\overrightarrow{A_i})$, which is quite typical, we just write $c_1(\overrightarrow{A_1})$.

We don't have an explicit notion of recursive types with which to type recursively defined data structures. However, as is well known, our constructor types together with (recursive) type constraints can capture the same notion implicitly. For example, our map function from Example 5.2 can be assigned the type:

$$\begin{array}{l} \mathsf{map} : \forall a\, b\, \ell_a\, \ell_b.\, C \Rightarrow (a \to b) \to \ell_a \to \ell_b \\ \qquad \text{where } C = \{\ \ell_a \equiv [] + (a :: \ell_a),\ \ell_b \equiv [] + (b :: \ell_b)\ \} \end{array}$$

This type says that map takes a function from a to b and a list of a and returns a list of b. Intuitively, the type 'list of a' is described by the type variable ℓ_a under the constraint that $\ell_a \equiv [] + (a :: \ell_a)$ and similarly for 'list of b' with ℓ_b constrained so that $\ell_b \equiv [] + (b :: \ell_b)$.

[3] Actually the one-step relation is parametrised by a module $\mathcal{M}$, but we leave this implicit.

$$\text{(IdS)}\ \frac{}{C,\ A \sqsubseteq B \vdash A \sqsubseteq B} \qquad \text{(TrS)}\ \frac{C \vdash A_1 \sqsubseteq A_2 \qquad C \vdash A_2 \sqsubseteq A_3}{C \vdash A_1 \sqsubseteq A_3}$$

$$\text{(ToS)}\ \frac{C \vdash A' \sqsubseteq A \qquad C \vdash B \sqsubseteq B'}{C \vdash A \to B \sqsubseteq A' \to B'} \qquad \text{(FrS)}\ \frac{C \vdash A \sqsubseteq A' \qquad C \vdash B' \sqsubseteq B}{C \vdash A \rightarrowtail B \sqsubseteq A' \rightarrowtail B'}$$

$$\text{(OkS)}\ \frac{}{C \vdash A \sqsubseteq \mathsf{Ok}} \qquad \text{(SmS)}\ \frac{C \vdash A_{i,c} \sqsubseteq B_{i,c}\ (\forall c \in I, \forall i \in [1..n_c])}{C \vdash \Sigma_{c \in I}\, c(A_{1,c}, \ldots, A_{n_c,c}) \sqsubseteq \Sigma_{d \in J}\, d(B_{1,d}, \ldots, B_{n_b,b})}\ I \subseteq J$$

Fig. 3. Constrained subtyping.

In a constrained type system, it is usual to define subtyping with respect to a context containing subtyping formulas, and so we have the following.

Definition 5.4. A *subtyping judgement* is a triple $C \vdash A \sqsubseteq B$ in which C is a set of type constraints and $A \sqsubseteq B$ is a type constraint. Provability is defined using the rules of Figure 3. We extend the notion of provability to sets of constraints, writing $C \vdash C'$ just if $C \vdash A \sqsubseteq B$ for every $A \sqsubseteq B \in C'$.

The rule (IdS) allows for justification with respect to the context, and the rule (TrS) ensures closure under transitivity of subtyping. Rule (ToS) gives the familiar relationship between sufficiency arrow types and (FrS) describes the dual relationship between necessity arrow types. Rule (OkS) puts Ok at the top of the subtype ordering. Finally, the rule (SmS) allows for subtyping between sum types. It says that a sum type K_1 is a subtype of K_2 just if whenever $c(A_1, \ldots, A_n)$ is a summand of K_1, then there is a summand of shape $c(B_1, \ldots, B_n)$ in K_2 and, moreover, the argument types are covariantly related. For example, using the constructors from Example 5.2, we have $\vdash [] \sqsubseteq [] + (\mathsf{Ok} :: \mathsf{Ok})$ and $a \sqsubseteq b \vdash (a :: []) \sqsubseteq [] + (b :: [])$.

Finally, we have the two-sided type system itself. In the interests of making the syntactical soundness proof as smooth as possible (by making the system as close to a traditional type system as possible), we present the type system as an intuitionistic sequent calculus, in the sense that there will be allowed *at most one* formula on the right hand side. We choose not to have an explicit component in the judgement for constraints in order to simplify the notation.

Definition 5.5 (Typing Formulas and Type Assignment). A *typing formula* is a pair of shape either $M : A$ or $\mathsf{f} : S$, with M a term, A a monotype, f a top-level identifier and S a type scheme. A *typing judgment* of the system consists of a pair $\Gamma \vdash \Delta$ in which Γ is a finite set of typing and subtype formulas and Δ is a finite set of typing formulas of shape $M : A$ and whose size is at most 1. For brevity, by some abuse, we will write Γ even for the subset $\{A \sqsubseteq B \mid A \sqsubseteq B \in \Gamma\}$ of subtype constraints contained therein.

The rules of the type system are given in Figure 4. We additionally require that: (i) in the rules (AbsR), (AbnR), (MchL), (MchR) and (FixR), the bound variables in the principal subject of the conclusion do not occur in Γ or Δ; and (ii) in the rule (GVar), the vector of types $\vec{B}$ has the same length as the vector of type variables $\vec{a}$; and (iii) the rules (CnsK) and (CnsL) require that $n > 0$ and $1 \leq i \leq n$.

Many of the typing rules of Figure 4 are recognisable from Sections 2 and 3, so we will just comment on new aspects. First, we have separate typing rules for local variables (introduced by abstractions and pattern-match cases) (LVar), and top-level identifiers (GVar). Top-level identifiers can be assumed to have polymorphic type schemes $\forall\, \vec{a}\,.\, C \Rightarrow A$, so the (GVar) rule allows for the instantiation of quantified type variables $\vec{a}$ by a vector of monotypes $\vec{B}$, subject to the requirement

Structural

$$\text{(VarK)}\ \frac{}{\Gamma \vdash x : \mathsf{Ok}} \qquad \text{(GVar)}\ \frac{}{\Gamma,\ f : \forall\, \vec{a}\,.C \Rightarrow A \vdash f : A[\vec{B}/\vec{a}]}\ \Gamma \vdash C[\vec{B}/\vec{a}]$$

$$\text{(LVar)}\ \frac{}{\Gamma,\ x : A \vdash x : A} \qquad \text{(SubL)}\ \frac{\Gamma,\ M : B \vdash \Delta}{\Gamma,\ M : A \vdash \Delta}\ \Gamma \vdash A \sqsubseteq B \qquad \text{(SubR)}\ \frac{\Gamma \vdash M : B}{\Gamma \vdash M : A}\ \Gamma \vdash B \sqsubseteq A$$

Functions

$$\text{(AbsR)}\ \frac{\Gamma,\ x : B \vdash M : A}{\Gamma \vdash (\lambda x.\, M) : B \to A} \qquad \text{(AbnR)}\ \frac{\Gamma,\ M : A \vdash x : B}{\Gamma \vdash (\lambda x.\, M) : B \rightarrowtail A}$$

$$\text{(AppL)}\ \frac{\Gamma \vdash M : B \rightarrowtail A \qquad \Gamma,\ N : B \vdash \Delta}{\Gamma,\ M\,N : A \vdash \Delta} \qquad \text{(AppR)}\ \frac{\Gamma \vdash M : B \to A \qquad \Gamma \vdash N : B}{\Gamma \vdash M\,N : A}$$

Constructors

$$\text{(CnsL)}\ \frac{\Gamma,\ M_i : A_i \vdash \Delta}{\Gamma,\ c(M_1, \ldots, M_n) : c(A_1, \ldots, A_n) + K \vdash \Delta} \qquad \text{(CnsR)}\ \frac{\Gamma \vdash M_i : A_i\ (\forall i)}{\Gamma \vdash c(M_1, \ldots, M_m) : c(A_1, \ldots, A_m)}$$

Pattern Matching

$$\text{(MchR)}\ \frac{\Gamma \vdash M : \Sigma_{i=1}^{k}\, p_i[\overrightarrow{B_x/x}] \qquad \Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A\ (\forall i)}{\Gamma \vdash \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k}(p_i \mapsto P_i)\} : A}$$

$$\text{(MchL)}\ \frac{\Gamma,\ P_i : A \vdash x : B_x\ (\forall i. \forall x \in \mathsf{FV}(p_i)) \qquad \Gamma,\ (M, P_i) : (p_i[\overrightarrow{B_x/x}], A) \vdash \Delta\ (\forall i)}{\Gamma,\ \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k}(p_i \mapsto P_i)\} : A \vdash \Delta}$$

Fixpoints and Evaluation

$$\text{(FixR)}\ \frac{\Gamma,\ x : A \vdash M : A}{\Gamma \vdash \mathsf{fix}\, x.\, M : A} \qquad \text{(CnsK)}\ \frac{\Gamma,\ M_i : \mathsf{Ok} \vdash \Delta}{\Gamma,\ c(M_1, \ldots, M_n) : \mathsf{Ok} \vdash \Delta} \qquad \text{(FunK)}\ \frac{\Gamma,\ M : \mathsf{Ok} \rightarrowtail A \vdash \Delta}{\Gamma,\ M\,N : \mathsf{Ok} \vdash \Delta}$$

Disjointness

$$\text{(CnsDL)}\ \frac{}{\Gamma,\ c(M_1, \ldots, M_n) : A \vdash \Delta}\ \begin{array}{l} A \text{ is an arrow} \\ \text{or shape } \Sigma_{d \in I}\, d(\ldots) \text{ with } c \notin I \end{array} \qquad \text{(AbsDL)}\ \frac{}{\Gamma,\ \lambda x.\, M : K \vdash \Delta}$$

Fig. 4. Constrained type assignment.

that each of the constraints in C is already derivable from the type constraints assumed in Γ. The (CnsL), (CnsR) and (CnsK) rules allow for refuting and affirming the types of constructor-headed terms.

The rules (MchL) and (MchR) are used for typing pattern-matching on the left and right respectively. The (MchR) rule is relatively standard: one must choose a typing B_x for each pattern-bound variable x (recall from additional requirement (ii) of Definition 5.1, that we assume all bound variables throughout the pattern-match term to have distinct names), in such a way that the scrutinee M is inside the sum of the induced pattern types. These are obtained by taking each pattern case $c(x_1, \ldots, x_n)$ and replacing the free variables x_i by the corresponding type B_{x_i}, giving $c(B_{x_1}, \ldots, B_{x_n})$. Then one must show that every branch can guarantee an A.

In (MchL), one must first derive a necessary type B_x for each pattern-bound variable x, based on how it is used in the corresponding branch of the match. These types then give rise to a type for the scrutinee, as above. Then one must show that, for each branch i, either the desired conclusion Δ follows from the fact that the body of the branch P_i has type A, or already from the fact that the scrutinee has type $p_i[\overrightarrow{B_x/x}]$. This disjunction is encoded by asserting that the pair (M, P_i) has type $(p_i[\overrightarrow{B_x/x}], A)$ so as to maintain the invariant that at most one non-trivial typing is introduced in each premise. An example of how this is used is at the end of this subsection.

The (CnsDL) rules above implement disjointness reasoning about constructor types. We have eschewed a general disjointness rule in order to simplify type inference (otherwise we would need to infer disjointness constraints). Instead we have two particular instances of disjointness for constructors (CnsDL) and abstractions (AbsDL).

Definition 5.6 (Top-level Function Typing). Top-level functions are typed according to the rule:

$$\frac{\Gamma \cup C \vdash \mathcal{M}(\mathsf{f}) : A}{\Gamma \vdash \mathsf{f} : \forall\, \vec{a}\,.\, C \Rightarrow A}\, \vec{a} = \mathsf{FV}(C) \cup \mathsf{FV}(A)$$

We write $\vdash \mathcal{M} : \Gamma$ just if, (i) every top-level function of $\mathcal{M}$ appears as a subject of Γ, and (ii) every top-level function typing $\mathsf{f} : S \in \Gamma$ is properly justified $\Gamma \vdash \mathsf{f} : S$.

Consider the head function of Example 5.2. We show that head requires a cons with an element of type a as input in order to produce an a, i.e. $\mathsf{head} : \forall a.\, (a :: \mathsf{Ok}) \rightarrowtail a$. The derivation starts:

$$\text{(AbnR)}\ \frac{\mathsf{match}\ xs\ \mathsf{with}\ \{y :: ys \mapsto y\} : a \vdash xs : (a :: \mathsf{Ok})}{\vdash \lambda xs.\, \mathsf{match}\ xs\ \mathsf{with}\ \{y :: ys \mapsto y\} : (a :: \mathsf{Ok}) \rightarrowtail a}$$

Then, according to (MchL) we must derive types for the bound variables y and ys based on how they were necessarily used to obtain type a in their branch. In this case, their branch body is just y and so we can assign y the type a and ys, which is not used in the branch, must be given Ok. Thus the scrutinee must have type $a :: \mathsf{Ok}$. According to (MchL), we must show that the conclusion, $xs : (a :: \mathsf{Ok})$, either follows from this or from the body of the branch. In this case, it is clear that it follows already from the type of the scrutinee. This reasoning is captured as:

$$\text{(MchL)}\ \frac{\text{(LVar)}\ \dfrac{}{y : a \vdash y : a} \qquad \text{(VarK)}\ \dfrac{}{y : a \vdash ys : \mathsf{Ok}} \qquad \text{(CnsL)}\ \dfrac{\text{(LVar)}\ \dfrac{}{xs : (a :: \mathsf{Ok}) \vdash xs : (a :: \mathsf{Ok})}}{(xs, y) : (a :: \mathsf{Ok}, a) \vdash xs : (a :: \mathsf{Ok})}}{\mathsf{match}\ xs\ \mathsf{with}\ \{y :: ys \mapsto y\} : a \vdash xs : (a :: \mathsf{Ok})}$$

Consider the map function from Example 5.2. We will show that, when map is given a function that requires an a to produce a b, then to produce a list of b requires it be given a list of a:

$$\begin{aligned} \mathsf{map} : \forall a\, b\, \ell_a\, \ell_b.\, C \Rightarrow (a \rightarrowtail b) \rightarrow \ell_a \rightarrowtail \ell_b \\ \text{where } \{\ [] + (a :: \ell_a) \sqsubseteq \ell_a,\ \ell_b \sqsubseteq [] + (b :: \ell_b)\ \} \end{aligned}$$

The derivation begins as we have seen previously using (FixR) and (AbnR). The key part is to show that, under $\Gamma = C \cup \{f : a \succ\!\!- b,\ m : (a \succ\!\!- b) \to \ell_a \succ\!\!- \ell_b\}$:

$$\Gamma, \mathsf{match}\ xs\ \mathsf{with}\ \{[] \mapsto [] \mid y :: ys \mapsto f\ y :: m\ f\ ys\} : \ell_b \vdash xs : \ell_a$$

To use the (MchL) rule, we first derive types that were necessary for y and ys to produce a value of type ℓ_b in the cons branch (we omit some standard right-side reasoning in the latter).

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash f : a \succ\!\!- b}(\text{LVar}) \quad \dfrac{}{\Gamma, y : a \vdash y : a}(\text{LVar})}{\Gamma,\ f\ y : b \vdash y : a}(\text{AppL})}{\dfrac{\Gamma,\ (f\ y :: m\ f\ ys) : [] + (b :: \ell_b) \vdash y : a}{\Gamma,\ (f\ y :: m\ f\ ys) : \ell_b \vdash y : a}(\text{SubL})}(\text{CnsL}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash m\ f : \ell_a \succ\!\!- \ell_b} \quad \dfrac{}{\Gamma, ys : \ell_a \vdash ys : \ell_a}(\text{LVar})}{\Gamma,\ m\ f\ ys : \ell_b \vdash ys : \ell_a}(\text{AppL})}{\Gamma,\ (f\ y :: m\ f\ ys) : [] + (b :: \ell_b) \vdash y : \ell_a}(\text{SubL})}{\Gamma,\ (f\ y :: m\ f\ ys) : \ell_b \vdash ys : \ell_a}(\text{CnsL})$$

Then, we show that $xs : \ell_a$ is a necessary consequence of each branch being of type ℓ_b. In fact, it follows directly from the induced type of the scrutinee in each (we omit the (CnsL) at the root).

$$\dfrac{\dfrac{}{\Gamma,\ xs : (a :: \ell_a) \vdash xs : (a :: \ell_a)}(\text{LVar})}{\Gamma,\ xs : (a :: \ell_a) \vdash xs : \ell_a}(\text{SubR}) \qquad \dfrac{\dfrac{}{\Gamma,\ xs : [] \vdash xs : []}(\text{LVar})}{\Gamma,\ xs : [] \vdash xs : \ell_a}(\text{SubR})$$

As one final example, we show that map, given a function that requires an a to obtain a b, returns a cons with element of type b only if given a cons with element of type a, i.e:

$$\mathsf{map} : \forall a\, b.(a \succ\!\!- b) \to (a :: \mathsf{Ok}) \succ\!\!- (b :: \mathsf{Ok})$$

The key part of the derivation is again the (MchL) rule. Under $\Gamma = \{f : a \succ\!\!- b,\ m : (a \succ\!\!- b) \to (a :: \mathsf{Ok}) \succ\!\!- (b :: \mathsf{Ok})\}$, we find, as above, that the cons case requires y to be an a (left), but we don't require anything special of ys (right):

$$\dfrac{\vdots}{\Gamma,\ (f\ y :: m\ f\ ys) : \ell_b \vdash y : a} \qquad (\text{VarK})\ \dfrac{}{\Gamma,\ (f\ y :: m\ f\ ys) : \ell_b \vdash ys : \mathsf{Ok}}$$

When showing that the desired conclusion follows from the two cases, we need to use the choice offered by the pair $(M, P_i) : (p_i[\overrightarrow{B_x/x}], A)$ on the left of the second family of premises of (MchL). By using (CnsL) to choose one component of the pair with which to continue the proof, we can effectively ignore irrelevant cases. Here, the nil case is excluded by the type $b :: \mathsf{Ok}$ of the output (left), and the result follows from the type of the scrutinee in the cons case (abbreviating $M = f\ y :: m\ f\ ys$ and $\Delta = \{\ xs : (a :: \mathsf{Ok})\ \}$):

$$\dfrac{\dfrac{}{\Gamma,\ [] : (b :: \mathsf{Ok}) \vdash \Delta}(\text{CnsDL})}{\Gamma,\ (xs, []) : ([], b :: \mathsf{Ok}) \vdash \Delta}(\text{CnsL}) \qquad \dfrac{\dfrac{}{\Gamma,\ xs : (a :: \mathsf{Ok}) \vdash \Delta}(\text{LVar})}{\Gamma,\ (xs, M) : (a :: \mathsf{Ok}, b :: \mathsf{Ok}) \vdash \Delta}(\text{CnsL})$$

We can now prove the example of ill-typedness in the introduction, the term that takes the head of a list obtained by mapping over the empty list will not evaluate: $\mathsf{head}\ (\mathsf{map}\ (\lambda x.\, x)\ []) : \mathsf{Ok} \vdash$.

5.2 Constrained Type Inference

For the purpose of inferring types and, in the next part, proving soundness, we are going to restrict ourselves to judgements that are as close to those in a traditional type system as possible: they will have at most one typing whose subject is not a variable. Then the following definition is helpful:

Definition 5.7 (Type Environments). A finite set of formulas Γ is said to be a *type environment* just if (i) all the typing formulas contained therein are variable typings $x : A$ and top-level identifier typings $\mathsf{f} : A$, and (ii) if $M : A \in \Gamma$ and $M : B \in \Gamma$ then $A = B$. A type environment is said to be a *top-level type environment* just if all the typing formulas are top-level identifier typings.

Then, inference for the two-sided constrained type system works, broadly, like Hindley-Milner constrained type inference (see e.g. [Odersky et al. 1999]), but with two notable differences. The first is that most terms will have multiple 'principal' types. This is because there are several shapes of proof tree with the same subject in the conclusion, but whose inferred types are not necessarily related. Hence, type inference infers sets of types[4]. The other main difference is that our algorithm requires both the left environment Γ and the right environment Δ as input when inferring the types of a term on the left. Hence, inference is split into two procedures, InferL and InferR:

$$\mathsf{InferL} \in \mathcal{P}(\mathsf{Variable} \times \mathsf{Type}) \to \mathsf{Term} \to \mathcal{P}(\mathsf{Variable} \times \mathsf{Type}) \to \mathcal{P}(\mathsf{Judgment})$$
$$\mathsf{InferR} \in \mathcal{P}(\mathsf{Variable} \times \mathsf{Type}) \to \mathsf{Term} \to \mathcal{P}(\mathsf{Judgment})$$

The call $\mathsf{InferL}(\Gamma)(M)(\Delta)$ returns a finite set of judgements[5] $C \cup \Gamma,\ M : A \vdash \Delta$ that are principal in the sense that all provable judgements with the same left-environment Γ, subject M and right-environment Δ arise as a type-substitution instance of one of those in the set, but with a possibly stronger set of assumed type constraints. The call $\mathsf{InferR}(\Gamma)(M)$ acts similarly on the right.

The algorithms terminate because, modulo subtyping, every proof tree has a maximum height that is determined by the shape of the term. As usual, subtyping can be absorbed into the other rules in a so-called algorithmic system, and then inference consists of (implicitly) constructing the finite set of principal proof trees for the given term and environment(s). Inferring on the right for the term $\lambda x.\, \mathsf{f}\ x$ in the environment $\{\mathsf{f} : [] \rightarrowtail \mathsf{Ok}\}$, we will obtain:

$$\begin{array}{rcl}
\mathsf{Ok} \sqsubseteq a_1,\ a_1 \rightarrowtail a_2 \sqsubseteq a_3,\ \mathsf{f} : [] \rightarrowtail \mathsf{Ok} & \vdash & \lambda x.\, \mathsf{f}\ x : a_3 \\
\mathsf{Ok} \sqsubseteq a_1,\ [] \rightarrowtail \mathsf{Ok} \sqsubseteq a_3,\ a_3 \sqsubseteq a_2 \rightarrowtail a_1,\ a_1 \rightarrowtail a_4 \sqsubseteq a_5,\ \mathsf{f} : [] \rightarrowtail \mathsf{Ok} & \vdash & \lambda x.\, \mathsf{f}\ x : a_5 \\
a_2 \sqsubseteq a_1,\ [] \rightarrowtail \mathsf{Ok} \sqsubseteq a_3,\ a_3 \sqsubseteq a_2 \rightarrowtail a_4,\ a_1 \rightarrowtail a_4 \sqsubseteq a_5,\ \mathsf{f} : [] \rightarrowtail \mathsf{Ok} & \vdash & \lambda x.\, \mathsf{f}\ x : a_5 \\
a_1 \sqsubseteq a_2,\ [] \rightarrowtail \mathsf{Ok} \sqsubseteq a_3,\ a_3 \sqsubseteq a_2 \to a_4,\ a_1 \to a_4 \sqsubseteq a_5,\ \mathsf{f} : [] \rightarrowtail \mathsf{Ok} & \vdash & \lambda x.\, \mathsf{f}\ x : a_5
\end{array}$$

The first three correspond, modulo subtyping inferences, to proof trees rooted at (AbnR). The first corresponds to the case when the tree is then immediately closed by the (VarK) axiom. The second corresponds to proceeding by (AppL) before closing with (GVar) in the left branch and (VarK) in the right branch. The third is similar but corresponds to closing the right branch with (LVar) instead. The fourth corresponds to a tree rooted at (AbsR), after which the shape is completely determined (modulo subtyping). However, notice that the constraints returned in this last case are inconsistent, we have $[] \rightarrowtail \mathsf{Ok} \sqsubseteq a_3$ and $a_3 \sqsubseteq a_2 \to a_4$, but the former function type contains functions that go wrong after being given an input, whereas the latter function type does not. Indeed, one cannot assign a sufficiency arrow type to this term in the given environment under consistent subtyping assumptions. The full definition of type inference is given in Appendix B.

Theorem 5.8 (Correctness of the Type Inference Algorithm). *Let Γ, Δ be type environments without type constraints, C and C' be sets of constraints, M be a term, and A, A' be types. Then:*

(Left Soundness) *If $(C \cup \Gamma,\ M : A \vdash \Delta) \in \mathsf{InferL}(\Gamma)(M)(\Delta)$, then $C \cup \Gamma,\ M : A \vdash \Delta$ is provable.*

(Left Completeness) *If $C' \cup \Gamma,\ M : A \vdash \Delta$ is provable, then there exists a type substitution σ and a judgement $(C \cup \Gamma,\ M : A' \vdash \Delta) \in \mathsf{InferL}(\Gamma)(M)(\Delta)$ such that $A = A'\sigma$ and $C' \vdash C\sigma$.*

(Right Soundness) *If $(C \cup \Gamma \vdash M : A) \in \mathsf{InferR}(\Gamma)(M)$, then $C \cup \Gamma \vdash M : A$ is provable.*

(Right Completeness) *If $C' \cup \Gamma \vdash M : A$ is provable, then there exists a type substitution σ and a judgement $(C \cup \Gamma \vdash M : A') \in \mathsf{InferR}(\Gamma)(M)$ such that $A = A'\sigma$ and $C' \vdash C\sigma$.*

[4]One can observe the same phenomenon in, e.g. intersection type systems.

[5]We formalise it this way for convenience, but in practice it need only return C and A.

5.3 Syntactic Soundness

Rather than proving semantic soundness, we take the opportunity to show how one can prove a syntactic soundness result in the sense of Wright and Felleisen [1994], but first we need to generalise the definition of well-typed and ill-typed to account for top-level identifiers.

Definition 5.9. Suppose M is closed and Γ is a consistent top-level type environment with $\vdash \mathcal{M} : \Gamma$.

- We say that M is *well-typed in* Γ just if $\Gamma \vdash M : \mathsf{Ok}$.
- We say that M is *ill-typed in* Γ just if $\Gamma,\ M : \mathsf{Ok} \vdash$.

The force of the qualifier *consistent* is to require that the type constraints in a type environment are not contradictory. Several definitions are possible, and we use an adaptation of the syntactical definition given by Eifrig et al. [1995]. Since it is orthogonal to the two-sided aspect of the work, the definition has been relegated to Appendix C.

An appropriate formulation of *progress* for two-sided systems must take into account a non-trivial subject on the left of the turnstile as well as the right. On the right, as usual, we have that terms can either make a step or are already values. However, in a typing $M : A$ on the left, it is possible that M can make a step, already be stuck or even be a value. However, in the latter case, the value must not be in A. We can state this succinctly as follows:

Theorem 5.10 (Progress). *Suppose M is closed and Γ is a consistent, top-level type environment.*

- *If $\Gamma \vdash M : A$ then either M can make a step, or M is a value*
- *If $\Gamma,\ M : A \vdash$ then either M can make a step, or $\Gamma \not\vdash M : A$*

The formulation uses a fact about typing in our two-sided systems that is peculiar to systems with the necessity arrow: all closed values are typable on the right. We already remarked in Section 3 that every abstraction is typable on the right with $\mathsf{Ok} \rightarrowtail A$, and a simple induction shows that every constructor-headed term has a corresponding constructor-headed type. Thus, $\Gamma \not\vdash M : A$ above in particular implies that M is not a value of type A.

For *preservation*, it is typical to prove some preliminary lemmas that show closure under well-typed substitutions. In the two-sided case, there are many more of these substitution lemmas, since one has to take account of the possibilities that the substituted-for variable occurs on the opposite side of the turnstile to the subject (as usual), on the same side, or not at all (which also relies on the fact that closed values are typable). However, preservation then follows directly.

Theorem 5.11 (Preservation). *Suppose M is a closed term and Γ is a consistent, top-level type environment in which all top-level identifiers have justified typings $\vdash \mathcal{M} : \Gamma$.*

- *If $\Gamma \vdash M : A$ and $M \triangleright N$ then $\Gamma \vdash N : A$.*
- *If $\Gamma,\ M : A \vdash$ and $M \triangleright N$ then $\Gamma,\ N : A \vdash$.*

Finally, syntactic soundness follows immediately.

Theorem 5.12 (Syntactic Soundness). *Suppose M is a closed term and Γ is a consistent, top-level type environment in which all top-level identifiers have justified typings $\vdash \mathcal{M} : \Gamma$.*

- *If M is well-typed in Γ, then M does not go wrong.*
- *If M is ill-typed in Γ, then M does not evaluate.*

Proof. Suppose $\Gamma \vdash M : \mathsf{Ok}$ converges to a normal form N. By preservation, $\Gamma \vdash N : \mathsf{Ok}$ and, by progress, N is a value. Suppose $\Gamma,\ M : \mathsf{Ok} \vdash$ converges to a normal form N, then, by preservation, $\Gamma,\ N : \mathsf{Ok} \vdash$ and, by progress, $\Gamma \not\vdash N : \mathsf{Ok}$. Since closed values are well typed, N is not a value. □

It's interesting to note that we didn't need to restrict to finitely-verifiable types on the right of necessity arrow in order to achieve syntactic soundness. On *closed* terms, our system is unable to

say much about necessity with higher-types. For example, judgements of the shape on the left are only provable in the constrained system when $A = \mathsf{Ok}$, whereas the one on the right, involving an open subject, would be provable in the system of Sections 2 and 3:

$$\vdash \lambda x.\, \lambda y.\, M : A \rightarrowtail (B \to C) \qquad\qquad \vdash \lambda x.\, y : A \rightarrowtail (B \to C),\ y : B \to C$$

6 COMPLEMENTS AND THE SUCCESS SEMANTICS

The two-sided type systems of this paper provide a form of negation at the meta level: to establish $M \notin \mathcal{T}[\![A]\!]$ we can prove $M : A \vdash$. It is natural to wonder if one can internalise the negation by some operator. Of course, we can add logical negation $\neg$ at the level of formulas, so that we have $\vdash \neg(M : A)$ iff $M : A \vdash$, but allowing for compound formulas is a more significant departure from the world of traditional type systems. It would be neater to add a complement operator $-^c$ at the level of types, with the semantics defined in such a way that $\vdash M : A^c$ iff $M : A \vdash$. However, the asymmetry in the meaning of typing creates difficulties, as we now show.

We return to the simpler language and type system of Sections 2 and 3 for our investigation into complements and negation. However, we will only prove syntactic soundness, and so, like in the system of Section 5, we allow ourselves unrestricted necessity.

Definition 6.1. We extend the types in Definition 2.3 by a complement operator and by allowing arbitrary types on the right of necessity. $A ::= \cdots \mid A \rightarrowtail B \mid A^c$.

$$\text{(CompL)}\ \frac{\Gamma \vdash M : A,\ \Delta}{\Gamma,\ M : A^c \vdash \Delta} \qquad \text{(CompR)}\ \frac{\Gamma,\ M : A \vdash \Delta}{\Gamma \vdash M : A^c,\ \Delta}$$

The semantics of the new operator is to take the complement of the type with respect to the set of closed values. So, we extend the equations of Definition 4.1 by $[\![A^c]\!] = \mathsf{Vals}_0 \setminus [\![A]\!]$. We also posit the new typing rules on the left.

Rule (CompL), which is closely related to our disjointness rule, is sound for our call-by-value semantics, but rule (CompR) is *unsound*. The problem is that when $M : A^c$ is not satisfied on the right by some θ, it does not imply that $M : A$ will necessarily be satisfied by θ on the left.

For example, consider the derivation to the right. If this were sound in the call-by-value semantics, $\mathsf{pred}(\underline{\mathsf{id}})$ would be guaranteed to *diverge*, since we would have that $\mathsf{pred}(\underline{\mathsf{id}}) \in \mathcal{T}_\perp[\![\mathsf{Ok}^c]\!]$, and $[\![\mathsf{Ok}^c]\!] = \mathsf{Vals}_0 \setminus \mathsf{Vals}_0 = \emptyset$. This is clearly absurd, because $\mathsf{pred}(\underline{\mathsf{id}})$ goes wrong. Thus our semantics of Section 4 does not support (CompR).

$$\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{\vdash \underline{\mathsf{id}} : \mathsf{Nat} \to \mathsf{Nat}}\,\text{(Dis)}}{\underline{\mathsf{id}} : \mathsf{Nat} \vdash}\,\text{(OkPL)}}{\mathsf{pred}(\underline{\mathsf{id}}) : \mathsf{Ok} \vdash}\,\text{(CompR)}}{\vdash \mathsf{pred}(\underline{\mathsf{id}}) : \mathsf{Ok}^c}$$

6.1 The success semantics

The issue is that we have only the backward direction of the desired equivalence between the two sides of the typing judgement: $M \notin \mathcal{T}[\![A]\!]$ *iff* $M \in \mathcal{T}_\perp[\![A^c]\!]$. The forward direction fails because the antecedent can be true when M gets stuck, although this possibility is not afforded by consequent.

One possible remedy to this is to lift the complement operator to the level of computations (terms), in such a way that $M : A^c$ whether on the left or right really means $M \in \mathcal{T}[\![A]\!]^c = \mathsf{Terms}_0 \setminus \mathcal{T}[\![A]\!]$. Clearly, this version of complement acts like a real negation and thus the equivalence is obtained. However, a consequence of this is that one would have to give up on allowing complements nested inside of types, like $A^c \times B$, or else define the meaning of each shape of typing formula involving nested complements separately. More perniciously, one would be forced to track divergence very carefully, because it would no longer be the case that every formula on the right is satisfied by a term that diverges. Thus, the (Fix) rule would need to be carefully qualified by the kind of types with which it can be instantiated.

Our approach will instead be to weaken typing on the right of the judgement, so that a formula $M : A$ on the right can *always* be satisfied by a term that goes wrong.

Definition 6.2. The *success semantics* is defined as follows. First, we define, for each type A, the set of closed terms that may go wrong, diverge, or evaluate to a value in $[[A]]$.

$$\mathcal{T}_{\perp\!\!\!\not\downarrow}[[A]] = \{\ M \mid M \in \mathcal{T}[[A]] \text{ or } M \text{ diverges or } M \text{ goes wrong }\}$$

We redefine the meaning of the sufficiency arrow type to allow the body to go wrong when executing on an argument:

$$[[A \to B]] = \{\ \lambda x.\, M \in \mathsf{Vals}_0 \mid \forall V \in [[A]].\, M[V/x] \in \mathcal{T}_{\perp\!\!\!\not\downarrow}[[B]]\ \}$$

Finally, we redefine satisfaction on the right by saying that a formula $M : A$ is *satisfied by θ on the right* just if $M\theta \in \mathcal{T}_{\perp\!\!\!\not\downarrow}[[A]]$.

The success semantics has the strong point that the judgements are now symmetrical in the sense of the above equivalence. Thus both (CompL) and (CompR) are sound for the success semantics. In fact, we will show that all the typing rules presented in Sections 2 and 3 are (syntactically) sound for the success semantics too.

A considerable disadvantage of the success semantics is that we lose the true negatives theorem *well-typed programs don't go wrong*. Now, proving $\vdash M :$ Ok means that M may either evaluate, diverge or go wrong. In other words, it means nothing at all! We do however retain *ill-typed programs don't evaluate*, so this system is exclusively for proving that programs behave badly. This puts this system in the same territory as Erlang's celebrated *success types* [Lindahl and Sagonas 2006], which similarly provide no guarantees for terms that are well-typed.

However, the symmetry in the system allows for a considerable saving. Under the success semantics, $[[A \rightarrowtail B]] = [[B^c \to A^c]]$ – the conditions on membership in these types are the contrapositive of each other. Thus, there is the potential to simply define $B \rightarrowtail A$ as an abbreviation for $B^c \to A^c$. Then, the symmetry between (AppL) and (AppR), and the symmetry between (AbsR) and (AbnR) can be exploited to derive necessity from sufficiency + complement:

$$\dfrac{\dfrac{\Gamma \vdash M : B \rightarrowtail A,\ \Delta \qquad \dfrac{\Gamma,\ N : B \vdash \Delta}{\Gamma \vdash N : B^c,\ \Delta}\ (\text{CompR})}{\Gamma \vdash M\,N : A^c,\ \Delta}\ (\text{AppR})}{\Gamma,\ M\,N : A \vdash \Delta}\ (\text{CompL}) \qquad\qquad \dfrac{\dfrac{\dfrac{\Gamma,\ M : A \vdash x : B,\ \Delta}{\Gamma \vdash M : A^c,\ x : B,\ \Delta}\ (\text{CompR})}{\Gamma,\ x : B^c \vdash M : A^c,\ \Delta}\ (\text{CompL})}{\Gamma \vdash \lambda x.\, M : B \rightarrowtail A,\ \Delta}\ (\text{AbsR})$$

Thus, in the success system, the sufficiency arrow in combination with complements provides a complete treatment of the necessity arrow, and we could dispense with (AbnR) and (AppL) if desirable.

6.2 The one-sided system

However, we can go further than this, by exploiting a key symmetry of the typing rules that we have presented so far. Let us say that a typing formula $M : A$ is a *variable typing* just if M is a variable, and otherwise it is a *non-variable typing*. Let us say that an environment Γ is a *type environment* just in case every formula therein is a variable typing. In each of the rules we have introduced in Sections 2 and 3, and so far in 6, observe the following. If the conclusion has at most one non-variable typing formula, then each of the hypotheses will have at most one non-variable typing formula too. For example, one can see by inspection of Examples 2.6 and 2.7 of Section 2 that, in each judgement of the respective proof trees, at most typing formula is non-variable.

When a judgment contains at most one non-variable typing then, in the success semantics, it is equivalent to a judgement in which a non-variable typing is the only typing formula on the right-hand side of the turnstile. Thus, in the success type system, we are able to *normalise* the typing

$$\text{(Ok)}\ \frac{}{\Gamma \vdash M : \mathsf{Ok}} \qquad \text{(OkC1)}\ \frac{}{\Gamma,\, x : \mathsf{Ok}^c \vdash M : A} \qquad \text{(OkC2)}\ \frac{\Gamma \vdash M : \mathsf{Ok}^c}{\Gamma \vdash M : A}$$

$$\text{(Contra)}\ \frac{}{\Gamma,\, x : A,\, x : A^c \vdash M : A} \qquad \text{(Var)}\ \frac{}{\Gamma,\, x : A \vdash x : A} \qquad \text{(Disj)}\ \frac{\Gamma \vdash M : B}{\Gamma \vdash M : A^c}\, B \mid\mid A$$

$$\text{(Zero)}\ \frac{}{\Gamma \vdash \mathsf{zero} : \mathsf{Nat}} \qquad \text{(Succ1)}\ \frac{\Gamma \vdash M : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}(M) : \mathsf{Nat}} \qquad \text{(Succ2)}\ \frac{\Gamma \vdash M : \mathsf{Nat}^c}{\Gamma \vdash \mathsf{succ}(M) : A}$$

$$\text{(Pred1)}\ \frac{\Gamma \vdash M : \mathsf{Nat}}{\Gamma \vdash \mathsf{pred}(M) : \mathsf{Nat}} \qquad \text{(Pred2)}\ \frac{\Gamma \vdash M : \mathsf{Nat}^c}{\Gamma \vdash \mathsf{pred}(M) : A}$$

$$\text{(Abs)}\ \frac{\Gamma,\, x : B \vdash M : A}{\Gamma \vdash (\lambda x.\, M) : B \to A} \qquad \text{(Fix)}\ \frac{\Gamma,\, x : A \vdash M : A}{\Gamma \vdash \mathsf{fix}\, x.\, M : A} \qquad \text{(Let3)}\ \frac{\Gamma \vdash N : A}{\Gamma \vdash \mathsf{let}\ (x, y) = M\ \mathsf{in}\ N : A}$$

$$\text{(Let2)}\ \frac{\begin{array}{c}\Gamma \vdash M : (B_1 \times B_2)^c \\ \Gamma,\, x_i : B_i^c \vdash N : A\ (\forall i)\end{array}}{\Gamma \vdash \mathsf{let}\ (x, y) = M\ \mathsf{in}\ N : A} \qquad \text{(Let1)}\ \frac{\begin{array}{c}\Gamma \vdash M : B \times C \\ \Gamma,\, x_1 : B,\, x_2 : C \vdash N : A\end{array}}{\Gamma \vdash \mathsf{let}\ (x_1, x_2) = M\ \mathsf{in}\ N : A}$$

$$\text{(App1)}\ \frac{\Gamma \vdash M : B \to A \qquad \Gamma \vdash N : B}{\Gamma \vdash M\, N : A} \qquad \text{(App2)}\ \frac{\Gamma \vdash M : (\mathsf{Ok}^c \to A)^c}{\Gamma \vdash M\, N : A} \qquad \text{(App3)}\ \frac{\Gamma \vdash N : \mathsf{Ok}^c}{\Gamma \vdash M\, N : A}$$

$$\text{(Pair1)}\ \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash (M,\, N) : A \times B} \qquad \text{(Pair2)}\ \frac{\Gamma \vdash M_i : \mathsf{Ok}^c}{\Gamma \vdash (M_1, M_2) : A} \qquad \text{(Pair3)}\ \frac{\Gamma \vdash M_i : A_i^c}{\Gamma \vdash (M_1, M_2) : (A_1 \times A_2)^c}$$

$$\text{(IfZ1)}\ \frac{\Gamma \vdash M : \mathsf{Nat}^c}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P : A} \qquad \text{(IfZ2)}\ \frac{\Gamma \vdash N : A \qquad \Gamma \vdash P : A}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ P : A}$$

Fig. 5. One-Sided Type Assignment

rules by using (CompL) and (CompR) to exchange the positions of formulas between the two sides, until they form a traditional, one-sided type system. In fact, the use of complements allows us to eliminate further redundancies, and so we obtain a smaller system.

Definition 6.3 (One-sided type system). A *judgement* of the one-sided system is a pair $\Gamma \vdash M : A$ of a type environment Γ and a typing $M : A$. Provability is defined by the rules of Figure 5. In rules (Let1), (Let2), (Let3), (Abs) we require that the bound variables are not mentioned in Γ.

In many cases, the rules of Figure 5 derive from their two-sided counterparts simply by normalising the judgements involved, possibly by introducing complements. Rule (Disj) arises from (Dis) this way. Normalisation often introduces a complemented typing $M : A^c$ in the conclusion, but in most cases it is sound to generalise to $M : A$. An example is deriving (IfZ1) from (IfZL1).

The rule (IfZ2) arises in this way, and it subsumes both (IfZR) and (IfZL2). However, the soundness of this one-sided rule is not obvious at first glance. It is almost the same as the right side rule (IfZR), except that (IfZR) has a third premise. In (IfZ2) this premise is absent, and we need only show that each branch has type A in order to conclude that the whole conditional has type A – *whether or*

not the guard is of type Nat. However, this *is sound* for the success semantics, because if the guard normalises to something other than a numeral, then the conditional will go wrong, and thus satisfies any success type.

The rules (Ok), (OkC) and (Contra) represent structural features of the success semantics; namely that every term satisfies $\mathcal{T}_{\perp\!\!\!/}[[\mathsf{Ok}]]$, that no term satisfies $\mathcal{T}[[\mathsf{Ok}^c]]$ and that $\mathcal{T}[[A]] \cap \mathcal{T}[[A^c]]$ is empty for every type A. Rule (Contra) is necessary in order to account for the possibility that a two-sided judgment is provable due to a contradiction among the variable typings. For example, the two-sided judgment $x : \mathsf{Nat}, M : A \vdash x : \mathsf{Nat}$ is clearly provable using (Id), but when normalised to a one-sided judgement, it becomes $x : \mathsf{Nat}, x : \mathsf{Nat}^c \vdash M : A$ and (Var) is not applicable.

Theorem 6.4. *Suppose* Γ *and* Δ *are type environments.*

- *If* $\Gamma, M : A \vdash \Delta$ *in the two-sided system, then* $\Gamma \cup \Delta^c \vdash M : A^c$ *in the one-sided system.*
- *If* $\Gamma \vdash M : A, \Delta$ *in the two-sided system, then* $\Gamma \cup \Delta^c \vdash M : A$ *in the one-sided system.*

Example 6.5. Consider again the judgement $\vdash \underline{\mathsf{twice}} : (A \rightarrowtail A) \to A \rightarrowtail A$, which we proved as Example 2.7 of Section 2. As in that example, we use Γ to abbreviate $\{f : A \rightarrowtail A\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x : A^c \vdash f : A \rightarrowtail A}\,(\mathsf{Var}) \quad \dfrac{\dfrac{}{\Gamma, x : A^c \vdash f : A \rightarrowtail A}\,(\mathsf{Var}) \quad \dfrac{}{\Gamma, x : A^c \vdash x : A^c}\,(\mathsf{Var})}{\Gamma, x : A^c \vdash f\,x : A^c}\,(\mathsf{App})}{\Gamma, x : A^c \vdash f\,(f\,x) : A^c}\,(\mathsf{Var})}{\Gamma \vdash \lambda x.\, f\,(f\,x) : A \rightarrowtail A}\,(\mathsf{Abs})}{\vdash \lambda f x.\, f\,(f\,x) : (A \rightarrowtail A) \to A \rightarrowtail A}\,(\mathsf{Abs})$$

We prove syntactic soundness for the one-sided system using a progress and preservation argument. In fact, progress as usually understood is not necessary – since we allow going wrong on the right, there is no requirement to show that a subject in normal form is a value, merely that values have sensible types. We get the syntactic soundness of the two-sided system under the success semantics as a corollary via Theorem 6.4.

Let us say that a closed term M is *ill-typed* in the one-sided system just if $\vdash M : \mathsf{Ok}^c$.

Theorem 6.6 (One-Side Syntactic Soundness). *If* M *is ill-typed, then* M *does not evaluate.*

7 CONCLUSION AND RELATED WORK

We have introduced *two-sided type systems* which are sequent calculi for typing formulas, made possible through a new function type $B \rightarrowtail A$. We have shown several ways in which these calculi can be considered sound, and illustrated how left-sided rules can be added to a constrained type system. We also investigated the internalisation of negation, and a one-sided system without a true negatives theorem. Many of the basic constants of the programming languages we use today have necessary requirements on their inputs. However, traditional type systems largely ignore this basic aspect of their behaviour. Two-sided systems can use this additional dimension, and enable simple reasoning about program incorrectness as well as program correctness.

Typing contexts beyond variables. Many type systems for higher-order program verification generalise the typing context to allow logical formulae for the purpose of recording path conditions. For example, Liquid Types [Rondon et al. 2008; Vazou et al. 2015, 2013] and the systems of Terauchi [2010] and Unno and Kobayashi [2009] place the Boolean valued expression M (or an equivalent formula) into the context when proving that the 'then'-branch of a conditional, if M then N else P, is correctly typed. In a two-sided system, this can be managed very naturally by introducing separate types for true and false (as we have in our system of Section 5) and assuming $M : \mathsf{true}$. A rather

different use of the typing context was made by Curien and Herbelin [2000]. Their system is a kind of sequent calculus in which the formulas on the left of the turnstile comprise variable typings and a stack of terms (cf. Krivine machines), which is thought of as an evaluation context.

Constrained type systems. Constrained type systems and inclusion constraints go back at least to the work of Mitchell [1984]. A unified treatment is given by Odersky et al. [1999]. Our system takes inspiration from the one-sided system of Aiken et al. [1994] (see also Marlow and Wadler [1997]), though their constraints are far more expressive. Our approach to consistency of subtype constraints is an adaptation of the purely syntactic approach of Eifrig et al. [1995]. It is not clear how well our algorithm would scale in practice, ideally one would like some guarantees on the size of types or the efficiency of inference, such as features in more recent works by Dolan and Mycroft [2017] and Jones and Ramsay [2021]. None of these systems has a true positives theorem.

Success Typing. Our original motivation was to try to better understand the very effective *success typing* paradigm for Erlang [Lindahl and Sagonas 2006]. Work by Jakob and Thiemann [2015] provides a useful view through their falsification type system, in which one constructs a logical formula used to describe inputs that guarantee to crash the function. The formalisation of López-Fraguas et al. [2018] is also enlightening and they moreover extend the system with polymorphism.

Success Types are perhaps the best known example of a type-based true positives theorem, 'ill-typed programs always fail' [Sagonas 2010]. They are so named because they over-approximate the successes of expressions: if A is a success type and $M : A$ then M may succeed in evaluating to an A, but may also go wrong or diverge. Hence why we named our alternative semantics after this paradigm. Note, the 'ill-typed' of the slogan really means 'not well-typed', i.e. untypable.

A key feature is a bespoke function type. It is defined in [Lindahl and Sagonas 2006] as follows:

> *A success typing of a function f is a type signature $(\vec{\alpha}) \to \beta$, such that whenever an application $f(\vec{p})$ reduces to a value v, then $v \in \beta$ and $\vec{p} \in \vec{\alpha}$.*

Thus, the *success arrow*, let us write $\to_s$, can be understood as having the right rule shown to the side. That is, it contains a component of necessity, but restricted to Ok on the right. Hence, with only this arrow, one can refute that an application MN has a type A only when $A = \mathsf{Ok}$. Yet, despite this, success types appear to be highly effective in practice. One reason could be the following: if an application MN fails to evaluate to an A, then there are three possibilities, it could go wrong, it could diverge, or it could evaluate to something else. In the first two cases, one can attempt to refute that it has type Ok, and in the third, one can attempt to affirm that it has a type that is disjoint from A instead. Thus, it is likely that one can get quite far with only this form of necessity.

$$\frac{\Gamma,\, M : \mathsf{Ok} \vdash x : \mathsf{Ok} \qquad \Gamma,\, M : \mathsf{Ok} \vdash M : B}{\Gamma \vdash \lambda x.\, M : A \to_s B,\, \Delta}$$

Logics for incorrectness. Incorrectness logic of O'Hearn [2019] has sparked a new interest in systems for reasoning about program behaviour that enjoy true positives theorems. Subsequent work has extended its scope [Raad et al. 2022], and demonstrated its real-world effectiveness [Le et al. 2022]. A key feature of these logics is *under-approximation* as a means to achieve true positives. In incorrectness logic, states in a postcondition must be *reachable*, and this leads to the need to reason about termination. By contrast, our Theorem 4.5 showed our types to be safety properties. The foundation of our system is not under-approximation, but necessity. An incorrectness triple such as $[A]M[B]$ could be seen as analogous to a function typing $M : A \Rrightarrow B$ in which every value in the type B can be obtained by applying M to some input in A. By contrast, our necessity arrow requires that no B can be obtained except from an input in A. When $M : A \rightarrowtail B$ is provable, type B is an *over-approximation* of M when run on input A, and thus the usual methods of abstraction,

such as recursive procedure invariants, still apply. However, set against this, one cannot simply drop disjunctions from Δ and remain sound. A very interesting compromise along these lines is the Outcome Logic of Zilberstein et al. [2023]. Here there is a distinction between Boolean disjunction and outcome disjunction. The former has true as annihilator, but the latter does not, and this allows for the dropping of outcomes for efficiency.

Type systems with complement. A complement operator appears in the system of Aiken et al. [1994] and a negation in the work of Parreaux and Chau [2022], though neither has a true positives theorem. In the highly expressive dependent type system of Unno et al. [2017], none of the programs can go wrong, but the system is nevertheless very well-equipped for incorrectness reasoning, and includes sophisticated means for reasoning about recursion and tracking divergence. Since their system can express the complements of arbitrary types, it should be possible to encode a version of our necessity arrow. One can think of our one-sided system of Section 6 as a kind of sweet spot for partial correctness, in which the theory and automation become particularly straightforward.

REFERENCES

Alexander Aiken, Edward L. Wimmers, and T. K. Lakshman. 1994. Soft Typing with Conditional Types. In *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*. 163–173. https://doi.org/10.1145/174675.177847

Pierre-Louis Curien and Hugo Herbelin. 2000. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, Martin Odersky and Philip Wadler (Eds.). ACM, 233–243. https://doi.org/10.1145/351240.351262

Stephen Dolan and Alan Mycroft. 2017. Polymorphism, Subtyping, and Type Inference in MLsub. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (Paris, France) *(POPL '17)*. Association for Computing Machinery, New York, NY, USA, 60–72. https://doi.org/10.1145/3009837.3009882

Jonathan Eifrig, Scott Smith, and Valery Trifonov. 1995. Sound Polymorphic Type Inference for Objects. In *Proceedings of the Tenth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications* (Austin, Texas, USA) *(OOPSLA '95)*. Association for Computing Machinery, New York, NY, USA, 169–184. https://doi.org/10.1145/217838.217858

Robert Jakob and Peter Thiemann. 2015. A Falsification View of Success Typing. In *NASA Formal Methods*, Klaus Havelund, Gerard Holzmann, and Rajeev Joshi (Eds.). Springer International Publishing, Cham, 234–247. https://doi.org/10.1007/978-3-319-17524-9_17

Eddie Jones and Steven Ramsay. 2021. Intensional Datatype Refinement: With Application to Scalable Verification of Pattern-Match Safety. *Proc. ACM Program. Lang.* 5, POPL, Article 55 (jan 2021), 29 pages. https://doi.org/10.1145/3434336

Quang Loc Le, Azalea Raad, Jules Villard, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. 2022. Finding Real Bugs in Big Programs with Incorrectness Logic. *Proc. ACM Program. Lang.* 6, OOPSLA1, Article 81 (apr 2022), 27 pages. https://doi.org/10.1145/3527325

Tobias Lindahl and Konstantinos Sagonas. 2006. Practical Type Inference Based on Success Typings. In *Proceedings of the 8th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming* (Venice, Italy) *(PPDP '06)*. Association for Computing Machinery, New York, NY, USA, 167–178. https://doi.org/10.1145/1140335.1140356

Francisco J. López-Fraguas, Manuel Montenegro, and Gorka Su\'arez-Garc\'ia. 2018. Polymorphic success types for Erlang. In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning (EPiC Series in Computing, Vol. 57)*, Gilles Barthe, Geoff Sutcliffe, and Margus Veanes (Eds.). EasyChair, 515–533. https://doi.org/10.29007/w2m2

Simon Marlow and Philip Wadler. 1997. A Practical Subtyping System for Erlang. In *Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming* (Amsterdam, The Netherlands) *(ICFP '97)*. Association for Computing Machinery, New York, NY, USA, 136–149. https://doi.org/10.1145/258948.258962

John C. Mitchell. 1984. Coercion and Type Inference. In *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Salt Lake City, Utah, USA) *(POPL '84)*. Association for Computing Machinery, New York, NY, USA, 175–185. https://doi.org/10.1145/800017.800529

Martin Odersky, Martin Sulzmann, and Martin Wehr. 1999. Type Inference with Constrained Types. *TAPOS* 5, 1 (1999), 35–55. https://doi.org/10.1002/(SICI)1096-9942(199901/03)5:1<35::AID-TAPO4>3.0.CO;2-4

Peter W. O'Hearn. 2019. Incorrectness Logic. *Proc. ACM Program. Lang.* 4, POPL, Article 10 (dec 2019), 32 pages. https://doi.org/10.1145/3371078

Lionel Parreaux and Chun Yin Chau. 2022. MLstruct: Principal Type Inference in a Boolean Algebra of Structural Types. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 141 (oct 2022), 30 pages. https://doi.org/10.1145/3563304

Azalea Raad, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. 2022. Concurrent Incorrectness Separation Logic. *Proc. ACM Program. Lang.* 6, POPL, Article 34 (jan 2022), 29 pages. https://doi.org/10.1145/3498695

Patrick Maxim Rondon, Ming Kawaguchi, and Ranjit Jhala. 2008. Liquid types. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13, 2008.* 159–169. https://doi.org/10.1145/1375581.1375602

Konstantinos Sagonas. 2010. Using Static Analysis to Detect Type Errors and Concurrency Defects in Erlang Programs. In *Functional and Logic Programming*, Matthias Blume, Naoki Kobayashi, and Germán Vidal (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 13–18. https://doi.org/10.1007/978-3-642-12251-4_2

Tachio Terauchi. 2010. Dependent types from counterexamples. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010.* 119–130. https://doi.org/10.1145/1706299.1706315

Hiroshi Unno and Naoki Kobayashi. 2009. Dependent type inference with interpolants. In *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal.* 277–288. https://doi.org/10.1145/1599410.1599445

Hiroshi Unno, Yuki Satake, and Tachio Terauchi. 2017. Relatively Complete Refinement Type System for Verification of Higher-Order Non-Deterministic Programs. *Proc. ACM Program. Lang.* 2, POPL, Article 12 (dec 2017), 29 pages. https://doi.org/10.1145/3158100

Niki Vazou, Alexander Bakst, and Ranjit Jhala. 2015. Bounded refinement types. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015.* 48–61. https://doi.org/10.1145/2784731.2784745

Niki Vazou, Patrick Maxim Rondon, and Ranjit Jhala. 2013. Abstract Refinement Types. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings.* 209–228. https://doi.org/10.1007/978-3-642-37036-6_13

A.K. Wright and M. Felleisen. 1994. A Syntactic Approach to Type Soundness. *Information and Computation* 115, 1 (1994), 38–94. https://doi.org/10.1006/inco.1994.1093

Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 93 (apr 2023), 29 pages. https://doi.org/10.1145/3586045

A ADDITIONAL MATERIAL IN SUPPORT OF SECTION 4

First, we give an explicit characterisation of getting stuck.

Definition A.1 (Getting Stuck). We separate the values into different classes:

- Vals – all values
- FunV – those values of shape $\lambda x.\, M$
- NatV – those values of shape $\underline{n}$
- ProdV – those values of shape $(V,\, W)$

A term is said to be a *stuckex* if it is of one of the following forms:

- $\mathsf{succ}(V)$ or $\mathsf{pred}(V)$ with $V \notin \mathsf{NatV}$
- $V\, M$ with $V \notin \mathsf{FunV}$
- $\mathsf{if}\, V\, \mathsf{then}\, N\, \mathsf{else}\, P$ with $V \notin \mathsf{NatV}$
- $\mathsf{let}\, (x, y) = V\, \mathsf{in}\, M$ with $V \notin \mathsf{ProdV}$

A term is said to be *stuck* just if it has shape $\mathcal{E}[M]$ with M a stuckex. A term M is said to *get stuck* just if $M \rhd^* N$ with N stuck. It is easy to see that a term in normal form is either stuck or is a value. Hence, every term either diverges, gets stuck or evaluates.

Next, we have a series of lemmas building up some basic results about the combinatorics of reduction. The overall goal is to show that replacing a subterm by another that either reduces to the same normal form or diverges leads to an essentially similar outcome.

In the following lemma, we use the phrase 'M is not blocked by z' to mean that M does not have shape $\mathcal{E}[z]$ for any evaluation context $\mathcal{E}$.

Lemma A.2. *If $M[N/z] = \mathcal{E}[P]$ and M is not blocked by z, then there is $\mathcal{F}$ and P' such that $M = \mathcal{F}[P']$ and $\mathcal{E} = \mathcal{F}[N/z]$ and $P = P'[N/z]$.*

Proof. By induction on $\mathcal{E}$.

- Suppose $\mathcal{E} = \Box$ and M is not blocked on z. Then $M[N/z] = P$ so let $\mathcal{E}' = \Box$ and $P' = M$.
- Suppose $\mathcal{E}$ is of shape $\mathsf{succ}(\mathcal{E}')$ and M is not blocked by z, so $M[N/z]$ is of shape $\mathsf{succ}(\mathcal{E}'[P])$. Since M is not blocked by z, it must have shape $\mathsf{succ}(Q)$ for some Q again not blocked by z, also $Q[N/z] = \mathcal{E}'[P]$. Hence, it follows from the induction hypothesis that there is some $\mathcal{E}''$ and P' such that $P'[N/z] = P$ and $\mathcal{E}' = \mathcal{E}''[N/z]$. Hence, let $\mathcal{F} := \mathsf{succ}(\mathcal{E}'')$.
- Suppose $\mathcal{E}$ is of shape $\mathsf{pred}(\mathcal{E}')$, we reason analogously to the previous case.
- Suppose $\mathcal{E}$ is of shape $(\mathcal{E}', Q_2)$ and M is not blocked by z, so $M[N/z]$ is of shape $(\mathcal{E}'[P], Q_2)$. Since M is not blocked by z, it has shape (Q_1, Q_2') and Q_1 is not blocked by z and $Q_1[N/z] = \mathcal{E}'[P]$ and $Q_2'[N/z] = Q_2$. It follows by induction that there is $\mathcal{E}''$ and P' such that $\mathcal{E}''[N/z] = \mathcal{E}'$ and $P'[N/z] = P$. Hence, let $\mathcal{F} := (\mathcal{E}'',\, Q_2')$.
- Suppose $\mathcal{E} = (V, \mathcal{E}')$ and M is not blocked by z. So, $M[N/z]$ is of shape $(V, \mathcal{E}'[P])$. Since M is not blocked by z, M must be of shape (V', Q) and Q is again not blocked by z, $V'[N/z] = V$ and $Q[N/z] = \mathcal{E}'[P]$. Then it follows from the induction hypothesis that there are $\mathcal{E}''$ and P' such that $P'[N/z] = P$ and $\mathcal{E}''[N/z] = \mathcal{E}'$. Hence, let $\mathcal{F}$ be $(V',\, \mathcal{E}'')$ and the result is immediate.
- Suppose $\mathcal{E}$ is of shape $\mathsf{proj}_i(\mathcal{E}')$ and M is not blocked by z. Then $M[N/z]$ is of shape $\mathsf{proj}_i(\mathcal{E}'[P])$. Since M is not blocked by z, it must be that M has shape $\mathsf{proj}_i(Q)$ and $Q[N/z] = \mathcal{E}'[P]$ and Q is not blocked by z. Hence, it follows from the induction hypothesis that there are P' and $\mathcal{E}''$ such that $P'[N/z] = P$ and $\mathcal{E}''[N/z] = \mathcal{E}'$. Thus, let $\mathcal{F}$ be $\mathsf{proj}_i(\mathcal{E}'')$.
- Suppose $\mathcal{E}$ is of shape $\mathsf{if}\, \mathcal{E}'\, \mathsf{then}\, Q_1\, \mathsf{else}\, Q_2$ and M is not blocked by z. Then $M[N/z] = \mathsf{if}\, \mathcal{E}'[P]\, \mathsf{then}\, Q_1\, \mathsf{else}\, Q_2$. Since M is not blocked by z, it must be that M has shape $\mathsf{if}\, Q_0\, \mathsf{then}\, Q_1'\, \mathsf{else}\, Q_2'$ and $Q_0[N/z] = \mathcal{E}'[P]$, $Q_1'[N/z] = Q_1$, $Q_2'[N/z] = Q_2$. Therefore, it follows from the induction hypothesis that there is P' and $\mathcal{E}''$ such that $P'[N/z] = P$ and $\mathcal{E}''[N/z] = \mathcal{E}'$. Thus, take $\mathcal{F}$ to be $\mathsf{if}\, \mathcal{E}''\, \mathsf{then}\, Q_1'\, \mathsf{else}\, Q_2'$ and the result follows.

- Suppose $\mathcal{E}$ has shape $(\lambda x.\, Q)\, \mathcal{E}'$ and M is not blocked by z. Then $M[N/z] = (\lambda x.\, Q)\, \mathcal{E}'[P]$. Since M is not blocked by z, M must have shape $(\lambda x.\, Q')\, R$ and $Q'[N/z] = Q$ and $R[N/z] = \mathcal{E}'[P]$ and R is not blocked by z. Thus, it follows from the induction hypothesis that there is P' and $\mathcal{E}''$ such that $P'[N/z] = P$ and $\mathcal{E}''[N/z] = \mathcal{E}'$. Then let $\mathcal{F}$ be $(\lambda x.\, Q')\, \mathcal{E}''$ and the result follows.
- Suppose $\mathcal{E}$ is of shape $\mathcal{E}'\, Q$ and M is not blocked by z. Then $M[N/z] = \mathcal{E}'[P]\, Q$. Then, since M is not blocked by z, M must have shape $R\, Q'$ with R not blocked by z and $R[N/z] = \mathcal{E}'[P]$ and $Q'[N/z] = Q$. Then it follows from the induction hypothesis that there is P' and $\mathcal{E}''$ with $P'[N/z] = P$ and $\mathcal{E}''[N/z] = \mathcal{E}'$. Thus, take $\mathcal{F}$ to be $\mathcal{E}''\, Q'$ and the result follows.

□

Lemma A.3. *If $M[N/z] \rhd P$ then M is blocked by z or there is M' such that $M \rhd M'$ and $P = M'[N/z]$*

Proof. We prove the following by induction on $\mathcal{E}$.

For all $\mathcal{E}$, for all M, if $M[N/z]$ is of shape $\mathcal{E}[P_1]$ and $P_1 \rhd P_2$ and M is not blocked by z, then there is some M' such that $M \rhd M'$ and $M'[N/z] = \mathcal{E}[P_2]$.

- Suppose $\mathcal{E} = \square$. Then assume $M[N/z]$ decomposes to $\mathcal{E}[P_1]$ and $P_1 \rhd P_2$ and M is not blocked by z. Then we analyse cases on the axiom $P_1 \rhd P_2$.
  - (Beta). Then $M[N/z]$ is of shape $(\lambda x.\, Q)\, V$ and $P_2 = Q[V/z]$. Since M is not blocked by z, it must be that M has shape $(\lambda x.\, Q')\, V'$ with $Q = Q'[N/z]$ and $V = V'[N/z]$. So, let M' be $Q'[V'/x]$ and we have $M'[N/z] = (Q'[V'/x])[N/z] = (Q'[N/z])[V'[N/z]/x]$, as required.
  - (PZ). Then $M[N/z]$ is of shape $\mathsf{pred}(\underline{0})$ and $P_2 = \underline{0}$. Since M is not blocked by z, it must have shape $\mathsf{pred}(\underline{0})$, so let M' be $\underline{0}$.
  - (PS). Then $M[N/z]$ is of shape $\mathsf{pred}(\underline{k+1})$ and $P_2 = \underline{k}$. Since M is not blocked by z, it must have shape $\mathsf{pred}(\underline{k+1})$, so let M' be $\underline{k}$.
  - (IfZ). Then $M[N/z]$ is of shape $\mathsf{if}\ \underline{0}\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$ and $P_2 = Q_1$. Since M is not blocked by z, it must have shape $\mathsf{if}\ \underline{0}\ \mathsf{then}\ Q_1'\ \mathsf{else}\ Q_2'$ with $Q_i = Q_i'[N/z]$. So, let M' be Q_1' and the result follows immediately.
  - (IfS). Then $M[N/z]$ is of shape $\mathsf{if}\ \underline{n+1}\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$ and $P_2 = Q_2[\underline{n}/x]$. Since M is not blocked by z, M must have shape $\mathsf{if}\ \underline{n+1}\ \mathsf{then}\ Q_1'\ \mathsf{else}\ Q_2'$ and $Q_i = Q_i'[N/z]$. Let M' be $Q_2'[\underline{n}/x]$. Then $M'[N/z] = (Q_2'[\underline{n}/x])[N/z] = (Q_2'[N/z])[\underline{n}/x]$, as required.
  - (Fix). Then $M[N/z]$ is of shape $\mathsf{fix}\, x.\, Q$ and $P_2 = Q[\mathsf{fix}\, x.\, Q/x]$. Since M is not blocked by z, it follows that M is of shape $\mathsf{fix}\, x.\, Q'$ and $Q = Q'[N/z]$. Let M' be $Q'[\mathsf{fix}\, x.\, Q'/x]$. Then $M'[N/z] = (Q'[\mathsf{fix}\, x.\, Q'/x])[N/z] = (Q'[N/z])[\mathsf{fix}\, x.\, Q'[N/z]/x]$, as required.
  - (Proj$_i$). Then $M[N/z]$ is of shape $\mathsf{proj}_i(V_1, V_2)$ and $P = V_i$. Since M is not blocked by z, it follows that M has shape $\mathsf{proj}(V_1', V_2')$ with $V_i = V_i'[N/z]$. Then let M' be V_i' and the result follows immediately.
- Suppose $\mathcal{E}$ is of shape $\mathsf{succ}(\mathcal{E}')$ and $M[N/z]$ is of shape $\mathsf{succ}(\mathcal{E}'[P_1])$ and $P_1 \rhd P_2$ and M is not blocked by z. Thus M must have shape $\mathsf{succ}(Q)$ and Q is again not blocked by z, also $Q[N/z] = \mathcal{E}'[P_1]$. Hence, it follows from the induction hypothesis that there is some Q' such that $Q \rhd Q'$ and $Q'[N/z] = \mathcal{E}'[P_2]$. Hence, let M' be $\mathsf{succ}(Q')$.
- Suppose $\mathcal{E}$ is of shape $\mathsf{pred}(\mathcal{E}')$, then we reason analogously to the previous case.
- Suppose $\mathcal{E}$ is of shape $(\mathcal{E}', Q_2)$ and $M[N/z]$ is of shape $(\mathcal{E}'[P_1], Q_2)$ and $P_1 \rhd P_2$ and M is not blocked by z. Thus M has shape (Q_1, Q_2') and Q_1 is not blocked by z and $Q_1[N/z] = \mathcal{E}'[P_1]$ and $Q_2'[N/z] = Q_2$. It follows by induction that there is Q_1' and $Q_1 \rhd Q_1'$ and $Q_1'[N/z] = \mathcal{E}'[P_2]$. So, let M' be (Q_1', Q_2').

- Suppose $\mathcal{E}$ is of shape $(V, \mathcal{E}')$ and $M[N/z]$ is of shape $(V, \mathcal{E}'[P_1])$ and $P_1 \rhd P_2$ and M is not blocked by z. So, M must be of shape (V', Q) and Q is again not blocked by z, $V'[N/z] = V$ and $Q[N/z] = \mathcal{E}'[P_1]$. Then it follows from the induction hypothesis that there is Q' such that $Q \rhd Q'$ and $\mathcal{E}'[P_1] = Q'[N/z]$. Hence, let M' be (V', Q') and the result is immediate.
- Suppose $\mathcal{E}$ is of shape $\mathsf{proj}_i(\mathcal{E}')$ and $M[N/z]$ is of shape $\mathsf{proj}_i(\mathcal{E}'[P_1])$ and $P_1 \rhd P_2$ and M is not blocked by z. So, it must be that M has shape $\mathsf{proj}_i(Q)$ and $Q[N/z] = \mathcal{E}'[P_1]$ and Q is not blocked by z. Hence, it follows from the induction hypothesis that there is Q' and $Q \rhd Q'$ and $Q'[N/z] = \mathcal{E}'[P_2]$. Thus, let M' be $\mathsf{proj}_i(Q')$.
- Suppose $\mathcal{E}$ is of shape $\mathsf{if}\ \mathcal{E}'\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$, $M[N/z] = \mathsf{if}\ \mathcal{E}'[P_1]\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2$ and $P_1 \rhd P_2$ and M is not blocked by z. So, it must be that M has shape $\mathsf{if}\ Q_0\ \mathsf{then}\ Q_1'\ \mathsf{else}\ Q_2'$ and $Q_0[N/z] = \mathcal{E}'[P_1]$, $Q_1'[N/z] = Q_1$, $Q_2'[N/z] = Q_2$. Therefore, it follows from the induction hypothesis that there is Q_0' such that $Q_0 \rhd Q_0'$ and $Q_0'[N/z] = \mathcal{E}'[P_2]$. Thus, take M' to be $\mathsf{if}\ Q_0'\ \mathsf{then}\ Q_1'\ \mathsf{else}\ Q_2'$ and the result follows.
- Suppose $\mathcal{E}$ has shape $(\lambda x.\, Q)\, \mathcal{E}'$ and $M[N/z] = (\lambda x.\, Q)\, \mathcal{E}'[P_1]$ and $P_1 \rhd P_2$ and M is not blocked by z. So, M must have shape $(\lambda x.\, Q')\, R$ and $Q'[N/z] = Q$ and $R[N/z] = \mathcal{E}'[P_1]$ and R is not blocked by z. Thus, it follows from the induction hypothesis that there is R' such that $R \rhd R'$ and $R'[N/z] = \mathcal{E}'[P_2]$. Then let M' be $(\lambda x.\, Q')\, R'$ and the result follows.
- Suppose $\mathcal{E}$ is of shape $\mathcal{E}'\, Q$ and $M[N/z] = \mathcal{E}'[P_1]\, Q$ and $P_1 \rhd P_2$ and M is not blocked by z. Then M must have shape $R\, Q'$ with R not blocked by z and $R[N/z] = \mathcal{E}'[P_1]$ and $Q'[N/z] = Q$. Then it follows from the induction hypothesis that there is R' with $R \rhd R'$ and $R'[N/z] = \mathcal{E}'[P_2]$. Thus, take M' to be $R'\, Q'$ and the result follows.

□

Lemma A.4. *If $M[N/x] \rhd^* P$ with P a normal form and $Q \lesssim N$, then either $M[Q/x]$ diverges or there is some M' such that $P = M'[N/x]$ and $M[Q/x] \rhd^* M'[Q/x]$.*

Proof. By lexicographic induction on the length of the sequence and the number of free occurrences of x in M.

- Suppose $M[N/x] = P$. Then take $M' := M$.
- Suppose $M[N/x] \rhd^k P$ for some $k > 0$. If M is $\mathcal{E}[x]$, then $M[Q/x] = (\mathcal{E}[Q/x])[Q]$. If Q diverges then so does $M[Q/x]$. Otherwise, N and Q normalise to the same normal form, say N'. Hence, $M[N/x] \rhd^* (\mathcal{E}[N/x])[N']$ and $M[Q/x] \rhd^* (\mathcal{E}[Q/x])[N']$, then $(\mathcal{E}[N/x])[N'] \rhd^m P$ with $m \leq k$ and the number of free occurrences of x in $\mathcal{E}[N']$ is decreased by 1. Hence, it follows from the induction hypothesis that $(\mathcal{E}[Q/x])[N']$ either diverges or there is some M' such that $P = M'[N/x]$ and $(\mathcal{E}[Q/x])[N'] \rhd^* M'[Q/x]$. The result follows immediately. Otherwise, we may assume M'' is not blocked by z and we can apply Lemma A.3 to obtain some M'' such that $M \rhd M''$. Then $M''[N/x] \rhd^{k-1} P$, and it follows from the induction hypothesis that either $M''[P/x]$ diverges, or there is some M' such that $P = M'[N/x]$ and $M''[Q/x] \rhd^* M'[Q/x]$. Then the result follows immediately since $M[Q/x] \rhd M''[Q/x]$.

□

Lemma A.5. *If $M[N/z]$ reduces to a value V and $P \lesssim N$, then either $M[P/z]$ diverges or $M[P/z]$ reduces to a value W and all of the following are true:*

(a) if V does not contain an abstraction, then $W = V$.
(b) if V has shape (V_1, V_2), then W has shape $(W_1[P/z], W_2[P/z])$, $W_1[N/z] = V_1$ and $W_2[N/z] = V_2$.
(c) if V has shape $\lambda x.\, P$, then W has shape $\lambda x.\, Q[P/z]$ with $Q[N/z] = P$.

Proof. The proof is by induction on V. Spp $M[N/z] \rhd^* V$. Then $M[P/z]$ either diverges or there is some value U such that $V = U[N/z]$ and $M[P/z] \rhd^* U[P/z]$. If $U = z$, then $N = V$ and $M[P/z] \rhd^* P$. If P diverges, then $M[P/z]$ diverges, as required. Otherwise $P \rhd^* V$ and $W = V$, (a), (b) and (c) follow immediately. Otherwise $U \neq z$ and we proceed by inspecting the cases.

- If V is a variable x, then since $U \neq z$, x must be different from z and $U = x$.
- If V is $\mathtt{zero}$, then since $U \neq z$, $U = \mathtt{zero}$.
- If V is of shape $\mathtt{succ}(\underline{n})$, then since $U \neq z$, U has shape $\mathtt{succ}\, W'$, and $W'[N/z] = \underline{n}$. Therefore, it follows from the induction hypothesis that $W'[P/z]$ reduces to $\underline{n}$ and the conclusion follows.
- If V is of shape (V_1, V_2), then since $U \neq z$, U is of shape (M_1, M_2) and $M_1[N/z] = V_1$ and $M_2[N/z] = V_2$. Therefore, M_1 and M_2 are themselves values. Moreover, if V did not contain an abstraction, then neither do V_1 or V_2 and the induction hypothesis yields that $M_1[P/z] = V_1$ and $M_2[P/z] = V_2$. Thus $W = V$.
- If V is of shape $\lambda x.\, P$, then since $U \neq z$, U has shape $\lambda x.\, Q$, with $Q[N/z] = P$ as required.

□

Lemma A.6. *If $M[N/z]$ gets stuck and $P \lesssim N$, then either $M[P/z]$ diverges or $M[P/z]$ gets stuck.*

Proof. Suppose $M[N/z]$ gets stuck. Then $M[N/z] \rhd^* P$ with P stuck. It follows that there is some R such that $P = R[N/z]$ and $M[P/z]$ either diverges or reduces to $R[P/z]$. Then $R[N/z]$ has shape $\mathcal{E}[Q]$ for some stuckex Q. By the forgoing lemma, R has shape $\mathcal{F}[Q']$ with $\mathcal{F}[N/z] = \mathcal{E}$ and $Q'[N/z] = Q$. If $Q' = z$, then $Q = N$ and by $P \lesssim N$ either $R[P/z]$ diverges or $R[P/z] \rhd^* (\mathcal{F}[P/z])[N']$ for some stuckex N', and thus also gets stuck. Otherwise, $Q' \neq z$ and we show in each case that $Q'[P/z]$ is a stuckex.

- If Q is of shape x, then since $Q' \neq z$, it must be that $x \neq z$ and $Q' = x$.
- If Q is of shape $V\, R$ with $V \notin \mathsf{FunV}$, then since $Q' \neq z$, it must be that Q' has shape $R_1\, R_2$ with $R_1[N/z] = V$. Hence, it follows from the previous lemma that either $R_1[P/z]$ diverges, in which case $M[P/z]$ diverges, or $R_1[P/z]$ is also a value not in FunV.
- If Q is of shape $\mathtt{proj}_i(V)$ with $V \notin \mathsf{ProdV}$, then since $Q' \neq z$, it must be that Q' has shape $\mathtt{proj}_i(R)$ and $R[N/z] = V$. Hence, it follows from the previous lemma that either $R_1[P/z]$ diverges, in which case $M[P/z]$ diverges, or $R_1[P/z]$ is also a value not in ProdV.

The remaining cases are similar. □

A.1 Proof of Theorem 4.5

Now we show the first part of Theorem4.5.

Theorem A.7. *Let $M[N/z]$ be closed, $Q \lesssim N$.*

(i) If $M[N/z] \in \mathcal{T}_\perp[\![A]\!]$, then $M[Q/z] \in \mathcal{T}_\perp[\![A]\!]$.
(ii) If $M[N/z] \notin \mathcal{T}[\![F]\!]$, then $M[Q/z] \notin \mathcal{T}[\![F]\!]$.

Proof. The proof is by induction on A.

- When $A = \mathsf{Nat}$, we reason as follows.
  (i) Suppose $M[N/z] \in \mathcal{T}_\perp[\![\mathsf{Nat}]\!]$, so either $M[N/z]$ diverges or $M[N/z] \rhd^* \underline{n}$. Then it follows from Lemma A.5 that either $M[Q/z]$ diverges or $M[Q/z]$ evaluates to $\underline{n}$.
  (ii) Suppose $M[N/z] \notin \mathcal{T}[\![\mathsf{Nat}]\!]$, so either (i) $M[N/z] \rhd^* V$ and $V \notin [\![\mathsf{Nat}]\!]$ or (ii) $M[N/z]$ gets stuck or (iii) $M[N/z]$ diverges. In cases (i) it follows from Lemma A.5 that $M[Q/z]$ either diverges or reduces to the same kind of value. In case (ii), it follows from Lemma A.6 that $M[Q/z]$ either diverges or gets stuck. In case (iii), $M[Q/z]$ also diverges. Hence, $M[Q/z] \notin \mathcal{T}[\![\mathsf{Nat}]\!]$.

- When A is of shape $B \times C$:
  (i) Suppose $M[N/z] \in \mathcal{T}_\perp[[B \times C]]$, so either $M[N/z]$ diverges, or $M[N/z] \rhd^* (V_1, V_2)$ with $V_1 \in [[B]]$ and $V_2 \in [[C]]$. In the former case, also $M[Q/z]$ diverges. Otherwise, it follows from Lemma A.5 that either $M[Q/z]$ diverges or evaluates to $(W_1[Q/z], W_2[Q/z])$ with $W_1[N/z] = V_1$ and $W_2[N/z] = V_2$. Hence, we have $W_1[N/z] \in [[B]]$ and thus also $\in \mathcal{T}_\perp[[B]]$; similarly $W_2[N/x] \in \mathcal{T}_\perp[[C]]$. Consequently, it follows from the induction hypotheses that $W_1[Q/z] \in \mathcal{T}_\perp[[B]]$ and $W_2[Q/z] \in \mathcal{T}_\perp[[C]]$. If either diverge, then $M[Q/z]$ diverges, as required. Otherwise, $W_1[Q/z]$ evaluates to a value in $[[B]]$ and $W_2[Q/z]$ evaluates to a value in $[[C]]$, as required.
  (ii) Suppose $M[N/z] \notin \mathcal{T}[[B \times C]]$. Here we can further assume that B and C are finitely verifiable. So either (i) $M[N/z] \rhd^* V$ with $V \notin [[B \times C]]$ or (ii) $M[N/z]$ gets stuck, or (iii) $M[N/z]$ diverges. In case (i), it follows from Lemma A.5 that either $M[Q/z]$ diverges or it evaluates to V. In case (ii) it follows from Lemma A.6 that $M[Q/z]$ either diverges or gets stuck. In case (iii) it follows that $M[Q/z]$ diverges. Hence, $M[Q/z] \notin \mathcal{T}[[B \times C]]$.
- When A is of shape $B \to C$, we need only prove (i). Suppose $M[N/z] \in \mathcal{T}_\perp[[B \to C]]$, so either $M[N/z]$ diverges or $M[N/z] \rhd^* \lambda x.P$ with $P[V/x] \in \mathcal{T}_\perp[[C]]$ for all $V \in [[B]]$. In the former case, $M[Q/z]$ diverges. Otherwise, it follows from Lemma A.5 that either $M[Q/z]$ diverges or there is some Q' such that $M[Q/z] \rhd^* \lambda x.Q'[Q/z]$ and $Q'[N/z] = P$. Let $V \in [[B]]$. We have $(Q'[V/x])[N/z] = (Q'[N/z])[V/x] \in \mathcal{T}_\perp[[C]]$ (the rearrangement is possible since N and V are necessarily closed) and so it follows from the induction hypothesis that $(Q'[V/x])[Q/z] \in \mathcal{T}_\perp[[C]]$ too.
- When A is of shape $B \rightarrowtail C$, we need only prove (i). Suppose $M[N/z] \in \mathcal{T}_\perp[[B \rightarrowtail F]]$, so either $M[N/z]$ diverges or $M[N/z] \rhd^* \lambda x.P$ with $P[V/x] \notin \mathcal{T}[[F]]$ if $V \notin [[B]]$. In the former case, $M[Q/z]$ diverges. Otherwise, it follows from Lemma A.5 that either $M[Q/z]$ diverges or there is Q' such that $M[Q/z] \rhd^* \lambda x.Q'[Q/z]$ and $Q'[N/z] = P$. Let V be a value and $V \notin [[B]]$. Hence, $(Q'[V/x])[N/z] = (Q'[N/x])[V/x] \notin \mathcal{T}[[F]]$ and it follows from the induction hypothesis that therefore $(Q'[Q/z])[V/x] \notin \mathcal{T}[[F]]$ either.
- When A is of shape Ok we reason as follows. For (i) suppose $M[N/z] \in \mathcal{T}_\perp[[\mathrm{Ok}]]$, then we reason as above, according to whichever value $M[N/z]$ reduces to. For (ii) suppose $M[N/z] \notin \mathcal{T}[[\mathrm{Ok}]]$, then either $M[N/z]$ gets stuck or $M[N/z]$ diverges. It follows from Lemma A.6 that, in both cases, $M[N'/z] \notin [[T]]$ too.

□

And then the second part of Theorem 4.5.

Theorem A.8. *For all types A and finitely verifiable types F:*

(i) If $C[\mathsf{fix}\, x.M] \notin \mathcal{T}_\perp[[A]]$, then there is some n_0, such that for all $n \geq n_0$: $C[\mathsf{fix}^n x.M]$.
(ii) If $C[\mathsf{fix}\, x.M] \in \mathcal{T}[[F]]$, then there is some n_0, such that for all $n \geq n_0$: $C[\mathsf{fix}^n x.M]$.

Proof. The proof is by induction on A.

- When A is Nat, we reason as follows.
  (i) Suppose $C[\mathsf{fix}\, x.M] \notin \mathcal{T}_\perp[[A]]$. Either $C[\mathsf{fix}\, x.M]$ evaluates to a value that is not a numeral or $C[\mathsf{fix}\, x.M]$ crashes. In both cases, it requires only finitely many reduction steps and, among them, only finitely many applications of fixpoint reduction, say k. Then we can take $n_0 := k$.
  (ii) Suppose $C[\mathsf{fix}\, x.M] \in \mathcal{T}[[A]]$. Then $C[\mathsf{fix}\, x.M]$ evaluates to a numeral, in finitely many steps. Among them, a certain number, say k, are applications of the fixpoint reduction. Therefore, for any $m \geq k$, $C[\mathsf{fix}^m x.M] \in \mathcal{T}[[A]]$.
- When A is Ok, we reason analogously.

- When A is of shape $B \times C$ we reason as follows.
  (i) Suppose $C[\mathsf{fix}\, x.\, M] \notin \mathcal{T}_\perp[\![B \times C]\!]$. Then either (i) $C[\mathsf{fix}\, x.\, M]$ evaluates to a value that is not a pair, or (ii) $C[\mathsf{fix}\, x.\, M]$ crashes, or (iii) $C[\mathsf{fix}\, x.\, M]$ evaluates to a pair value (V_1, V_2), but either $V_1 \notin [\![B]\!]$ or $V_2 \notin [\![C]\!]$. In all cases, the reduction sequences involved are finite, and so contain only finitely many applications of the fixpoint reduction. Hence, we may obtain the required bound as above. (i) and (ii) we can obtain a bound as above. In case (iii), first observe that this evaluation used only finitely many applications of the fixpoint reduction, say k_0, and so for any $k \geq k_0$, $C[\mathsf{fix}^k x.\, M]$ will reduce to a pair value (W_1, W_2). Now let us assume that it is $V_1 \notin [\![B]\!]$ because the other case is symmetrical. Note that, since V_1 is a value, we know that $V_1 \notin \mathcal{T}_\perp[\![B]\!]$ Next, observe that we can write V_1 as $D[\mathsf{fix}\, x.\, M]$ in the manner described above, and it follows from the induction hypothesis that there is m_0 and for all $m \geq m_0$, $D[\mathsf{fix}^m x.\, M] \notin \mathcal{T}_\perp[\![B]\!]$. Since each $D[\mathsf{fix}^m x.\, M]$ is a value, it follows that $D[\mathsf{fix}^m x.\, M] \notin [\![B]\!]$. Hence, for all $n \geq m_0 + k_0$, $C[\mathsf{fix}^n x.\, M] \notin \mathcal{T}_\perp[\![B \times C]\!]$.
  (ii) In this case, we may assume that B and C are finitely verifiable. Suppose $C[\mathsf{fix}\, x.\, M] \in \mathcal{T}[\![B \times C]\!]$. Then $C[\mathsf{fix}\, x.\, M]$ evaluates to a pair (V_1, V_2) (and this pair must not contain any abstraction, since B and C are finitely verifiable). Then this reduction sequence is finite, and applies the fixpoint reduction rule only a certain number, say n_0 of times. We can use this as the bound required by the conclusion.
- When A is $B \to C$, we need only consider (i). So suppose $C[\mathsf{fix}\, x.\, M] \notin \mathcal{T}_\perp[\![B \to C]\!]$. Then either (i) $C[\mathsf{fix}\, x.\, M]$ evaluates to a value that is not an abstraction, or (ii) $C[\mathsf{fix}\, x.\, M]$ crashes, or (iii) $C[\mathsf{fix}\, x.\, M]$ evaluates to an abstraction $\lambda y.\, P$, but there is a value $V \in [\![B]\!]$ such that $P[V/y] \notin \mathcal{T}_\perp[\![C]\!]$. In cases (i) and (ii) we can reason as above to obtain the bound. In case (iii), we can first observe that evaluating to an abstraction requires only finitely many applications of the fixpoint reduction rule, say m_0. So, for any $m \geq m_0$, $C[\mathsf{fix}^m x.\, M]$ will also evaluate to an abstraction, say $\lambda y.\, Q$. Then we note that we can write $P[V/y]$ as $D[\mathsf{fix}\, x.\, M]$, in which all descendants of occurrences of $\mathsf{fix}\, x.\, M$ in $C[\mathsf{fix}\, x.\, M]$ are replaced by holes in D (for example, by introducing labelled subterms). Then it follows from the induction hypothesis that there is some p_0 such that, for all $p \geq p_0$, $D[\mathsf{fix}^p x.\, M] \notin \mathcal{T}_\perp[\![C]\!]$. Hence, by construction, we have that, for all $n \geq m_0 + p_0$, $C[\mathsf{fix}^n x.\, M] \notin \mathcal{T}_\perp[\![B \to C]\!]$.
- When A is $B \rightarrowtail F$, we need only consider (i). So suppose $C[\mathsf{fix}\, x.\, M] \notin \mathcal{T}_\perp[\![B \rightarrowtail F]\!]$. Then either (i) $C[\mathsf{fix}\, x.\, M]$ evaluates to a value that is not an abstraction, or (ii) $C[\mathsf{fix}\, x.\, M]$ crashes, or (iii) $C[\mathsf{fix}\, x.\, M]$ evaluates to an abstraction $\lambda y.\, P$, but there is a value V such that $V \notin [\![B]\!]$ and yet $P[V/y] \in \mathcal{T}[\![F]\!]$. In cases (i) and (ii) we can find the bound as above. In case (iii), we first observe that evaluating to an abstraction requires only finitely many applications of the fixpoint reduction, say m_0. So, for any $m \geq m_0$, $C[\mathsf{fix}^m x.\, M]$ will also evaluate to an abstraction. Next, note that we can write $P[V/y]$ as $D[\mathsf{fix}\, x.\, M]$ by factoring out all descendants of $\mathsf{fix}\, x.\, M$ in the original term. Then, it follows from the induction hypothesis that there is some k_0 such that, for all $k \geq k_0$, $D[\mathsf{fix}^k x.\, M] \in \mathcal{T}[\![F]\!]$. Then, it follows from this construction that, for all $n \geq m_0 + k_0$, $C[\mathsf{fix}^n x.\, M] \notin \mathcal{T}_\perp[\![B \rightarrowtail F]\!]$.

□

Note, the contrapositive of part (i) of this result gives us the following principle: if $C[\mathsf{fix}^n x.\, M] \in \mathcal{T}_\perp[\![A]\!]$ for all n, then $C[\mathsf{fix}\, x.\, M] \in \mathcal{T}_\perp[\![A]\!]$.

Theorem 4.5. follows from Theorems A.7 and A.8 by observing that, since N and P are closed in the statement, we can always find some fresh variable z in order to write $C[N]$ as $(C[z])[N/z]$.

A.2 Proof of Theorem 4.6

Proof. Suppose $\lambda x.M \in [[A \to A]]$. Then, by definition, for all $V \in [[A]]$, $M[V/x] \in \mathcal{T}_\perp[[A]]$. It follows from Theorem 4.5 (i) that, (*) for all $N \in \mathcal{T}_\perp[[A]]$, $M[N/x] \in \mathcal{T}_\perp[[A]]$ (such N either diverge or evaluate to a value in $[[A]]$). We show that $\mathsf{fix}\, x.M \in \mathcal{T}_\perp[[A]]$ using the above principle, by induction on $n \in \mathbb{N}$.

- When $n = 0$, $\mathsf{fix}^n x.M = \underline{\mathsf{div}}$ and $\underline{\mathsf{div}} \in \mathcal{T}_\perp[[A]]$ by definition.
- When n is of shape $k+1$, we assume $\mathsf{fix}^k x.M \in \mathcal{T}_\perp[[A]]$. Then it follows from (*) that $M[\mathsf{fix}^k x.M/x] \in \mathcal{T}_\perp[[A]]$. This just means that $\mathsf{fix}^{k+1} x.M \in \mathcal{T}_\perp[[A]]$, as required.

□

A.3 Proof of Theorem 4.3

Finally, we can show semantic soundness, Theorem 4.3.

Proof. The proof is by induction on the derivation.

(Id) Suppose θ satisfies Γ, $x : A$ on the left, then $x\theta \in \mathcal{T}[[A]]$. Hence, $x\theta \in \mathcal{T}_\perp[[A]]$ and the conclusion follows.

(Dis) Suppose $A \,||\, B$ and that θ satisfies Γ, $M : A$ on the left, so that $M\theta \in \mathcal{T}[[A]]$. It follows that $M\theta \notin \mathcal{T}_\perp[[B]]$ (*). Then it follows from the induction hypothesis that θ satisfies $M : B$, Δ on the right, but by (*), θ does not satisfy $M : B$ on the right. Therefore, θ must satisfy Δ on the right, as required.

(ZeroR) By definition $\mathtt{zero}\theta = \mathtt{zero} \in \mathcal{T}_\perp[[\mathsf{Nat}]]$.

(SuccR) Suppose θ satisfies Γ on the left, then it follows from the induction hypothesis that θ satisfies $M : \mathsf{Nat}\ \Delta$ on the right. Suppose θ does not satisfy Δ on the right. Then θ satisfies $M : \mathsf{Nat}$ on the right, so either $M\theta$ diverges or $M\theta$ evaluates to a number. Hence θ satisfies $\mathtt{succ}(M) : \mathsf{Nat}$ on the right.

(PredR) Analogous to the previous case.

(FixR) Suppose $\Gamma,\ x : A \models M : A,\ \Delta$ and let θ be a valuation satisfying Γ on the left. To see that θ will satisfy $\mathsf{fix}\, x.M : A,\ \Delta$ on the right we suppose θ does not satisfy Δ on the right, and show that θ satisfies $\mathsf{fix}\, x.M : A$ on the right, i.e. $\mathsf{fix}\, x.M\theta \in \mathcal{T}_\perp[[A]]$. By this assumption and our original supposition, we have that, for all $V \in [[A]]$, $(M\theta)[V/x] \in \mathcal{T}_\perp[[A]]$, so $\lambda x.M\theta \in [[A \to A]]$. Then the result follows since typing is closed under taking fixpoints.

(LetR) Suppose θ satisfies Γ on the left, then it follows from the induction hypotheses that θ satisfies $M : B \times C,\ \Delta$ on the right. Suppose θ does not satisfy Δ. Hence, θ satisfies $M : B \times C$ on the right so $M\theta$ either diverges or evaluates to a pair (V, W) with $V \in [[B]]$ and $W \in [[C]]$. In the former case, the let expression diverges under θ and we are done. Otherwise $\theta \cup [V/x, W/y]$ satisfies $\Gamma,\ x : B,\ x : C$ on the left and it follows from the induction hypothesis that θ satisfies $N : A$ on the right, so $(N[V/x, W/y])\theta \in \mathcal{T}_\perp[[A]]$. The conclusion follows since $(\mathsf{let}\,(x, y) = M\,\mathsf{in}\,N)\theta \rhd^* (N[V/x, W/y])\theta$.

(AppR) Suppose θ satisfies Γ on the left. Then it follows from the induction hypotheses that θ satisfies $M : B \to A,\ \Delta$ and $N : B,\ \Delta$ on the right. Suppose θ does not satisfy Δ. Then $M\theta \in \mathcal{T}_\perp[[B \to A]]$ and $N\theta \in \mathcal{T}_\perp[[B]]$. If either diverges, then so does $(M\,N)\theta$ and we are done. Otherwise, $M\theta$ evaluates to some abstraction $\lambda x.P \in [[B \to A]]$ and $N\theta$ evaluates to a value V in $[[B]]$ and hence it follows from the definition that $(M\,N)\theta \rhd^* (\lambda x.P)\ V \rhd P[V/x] \in \mathcal{T}_\perp[[A]]$, as required.

(PairR) Suppose θ satisfies Γ on the left, then it follows from the induction hypotheses that θ satisfies $M : A,\ \Delta$ and $N : B,\ \Delta$ on the right. Suppose θ does not satisfy Δ on the right, then $M\theta \in \mathcal{T}_\perp[[A]]$ and $N\theta \in \mathcal{T}_\perp[[B]]$. If either diverge, then so does $(M, N)\theta$ and the conclusion is

immediate. Otherwise, $M\theta$ evaluates to some $V \in [[A]]$ and $N\theta$ evaluates to some $W \in [[B]]$ and thus $(M, N)\theta$ evaluates to $(V, W) \in [[A \times B]]$.

(AbsR) Suppose θ satisfies Γ on the left. Suppose θ does not satisfy Δ on the right, so we need to show that θ satisfies $(\lambda x.\, P) : B \to A$ on the right. So let $V \in [[B]]$. Then $\theta \cup [V/x]$ satisfies Γ, $x : B$ on the left, and the induction hypothesis gives that $P\theta \in \mathcal{T}_\perp[[A]]$ as required.

(AbnR) Suppose θ satisfies Γ on the left. Suppose θ does not satisfy Δ on the right, so we need to show that θ satisfies $(\lambda x.\, P) : B \rightarrowtail A$ on the right. So let V be a closed value and assume $(P\theta)[V/x] \in \mathcal{T}[[B]]$. Then $\theta \cup [V/x]$ satisfies Γ, $M : B$ on the left. Hence, it follows from the induction hypothesis that $\theta \cup [V/x]$ satisfies $x : A$ on the right, and so $V \in \mathcal{T}_\perp[[A]]$. Since V is a value, the conclusion is immediate.

(IfZR) Suppose θ satisfies Γ on the left. Then it follows from the induction hypothesis that θ satisfies (i) M : Nat, Δ, (ii) $N : A$, Δ and (iii) $P : A$, Δ on the right. Suppose that θ does not satisfy Δ. Then $M\theta$ either diverges or evaluates to a numeral, say $\underline{n}$. In the former case, the if expression diverges too and we conclude. Otherwise, if $n = 0$, it follows that if $M\theta$ then $N\theta$ else $P\theta \rhd^* N\theta$ and the result follows from (ii). If $n > 0$, then if $M\theta$ then $N\theta$ else $P\theta \rhd^* P\theta$ and the result follows from (iii).

(AppL) Suppose θ satisfies Γ, $(M\,N) : A$ on the left. Then $(M\,N)\theta \in \mathcal{T}[[A]]$. It follows from the induction hypothesis that θ satisfies $M : B \rightarrowtail A$, Δ on the right. If θ satisfies Δ on the right, then we are done. If θ satisfies $M : B \rightarrowtail A$ on the right, then $M\theta \in \mathcal{T}_\perp[[B \rightarrowtail A]]$. Since $(M\,N)\theta$ is guaranteed to terminate, it follows that $M\theta$ and $N\theta$ are too, and so $M\theta$ evaluates to some abstraction $\lambda x.\, P \in [[B \rightarrowtail A]]$ and $N\theta$ to some value V. It follows from the definition that since $(M\,N)\theta \rhd^* P[V/x] \in \mathcal{T}[[A]]$, $V \in [[B]]$ and hence $N\theta \in \mathcal{T}[[B]]$. Then θ satisfies Γ, $N : B$ on the left and the result follows from the induction hypothesis.

(LetL1) Suppose θ satisfies Γ, let $(x, y) = P$ in $N : A$ on the left, so let $(x, y) = P\theta$ in $N\theta \in \mathcal{T}[[A]]$. Therefore, it must be that $P\theta$ evaluates to a pair (V, W) and $(N\theta)[V/x, W/y] \in \mathcal{T}[[A]]$. Then $\theta \cup [V/x, W/y]$ satisfies Γ, $N : A$ on the left, and it follows from the induction hypothesis that it also satisfies Δ on the right. Since x and y are bound in the let, we may assume they do not occur free in Δ, so also θ satisfies Δ on the right.

(LetL2) Suppose θ satisfies Γ, let $(x, y) = P$ in $N : A$ on the left, so let $(x, y) = P\theta$ in $N\theta \in \mathcal{T}[[A]]$. It must be that $P\theta$ evaluates to some pair (V, W) and $(N\theta)[V/x, W/y] \in \mathcal{T}[[A]]$. Hence, $\theta \cup [V/x, W/y]$ satisfies Γ, $N : A$ on the left and it follows from the induction hypotheses that it also satisfies $x : B$, Δ and $y : C$, Δ on the right. If it satisfies Δ on the right, then θ satisfies Δ on the right since we mat assume the bound variables x and y do not occur in Δ. Otherwise, $V \in [[B]]$ and $W \in [[C]]$ and so we have that $P\theta \rhd^* (V, w) \in [[B \times C]]$. Therefore, it follows from the induction hypothesis that θ satisfies Δ on the right.

(SuccL) Suppose θ satisfies Γ, $\mathsf{succ}(P)$: Nat on the left, so $\mathsf{succ}(P\theta) \in \mathcal{T}[[\mathsf{Nat}]]$ and hence $P\theta$ must evaluate to a number. Then the conclusion follows from the induction hypothesis.

(PredL) Analogous to the previous case.

(IfZL1) Suppose θ satisfies Γ, if Q then N else $P : A$, so that if $Q\theta$ then $N\theta$ else $P\theta$ evaluates to an A. Then it must be that $Q\theta$ evaluates to a numeral, and then the result follows from the induction hypothesis.

(IfZL2) Suppose θ satisfies Γ, if Q then N else $P : A$, so that if $Q\theta$ then $N\theta$ else $P\theta$ evaluates to an A. Then it must be that $Q\theta$ evaluates to a numeral, say $\underline{n}$. If $n = 0$, then the if expression reduces to $N\theta \in \mathcal{T}[[A]]$ and the result follows from the first induction hypothesis. Otherwise, the if expression reduces to $P\theta$ and the result follows from the second induction hypothesis.

(OkVarR) It is immediate from the definition that $x\theta$ is a value.

(OkL) Suppose θ satisfies Γ, $M : A$ on the left. Then $M\theta \in \mathcal{T}[[A]]$. Since $\mathcal{T}[[A]] \subseteq \mathcal{T}[[\mathsf{Ok}]]$, the conclusion follows from the induction hypothesis.

(OkR) Symmetrical to the previous case.

(OkSL) Suppose θ satisfies Γ, $\mathsf{succ}(M) : \mathsf{Ok}$ on the left. So $\mathsf{succ}(M\theta)$ evaluates. The only values headed by succ are numerals, so it follows that $M\theta$ evaluates to a numeral and then the conclusion follows from the induction hypothesis.

(OkPL) Suppose θ satisfies Γ, $\mathsf{pred}(M) : \mathsf{Ok}$ on the left. So $\mathsf{pred}(M\theta)$ evaluates. Since $\mathsf{pred}(M\theta)$ can only reduce to a numeral, and this requires that $M\theta$ evaluate to a numeral, the conclusion follows from the induction hypothesis.

(OkPrL) Suppose θ satisfies Γ, $(M_1, M_2) : \mathsf{Ok}$. Then $(M_1\theta, M_2\theta)$ evaluates and it must be that it evaluates to a pair of values (V_1, V_2). Then it follows that each M_i evaluates and so the conclusion follows from the induction hypothesis.

(OkApL1) Suppose θ satisfies Γ, $M\,N : \mathsf{Ok}$ on the left, so $M\theta\,N\theta$ evaluates. Then it must be that $M\theta$ evaluates to an abstraction. Since, for any A, $[[\mathsf{Ok} \rightarrowtail A]]$ is the set of all abstractions, the conclusion follows from the induction hypothesis.

(OkApL2) Suppose θ satisfies Γ, $M\,N : \mathsf{Ok}$, so $M\theta\,N\theta$ evaluates. Then, in particular, $N\theta$ evaluates, and the conclusion follows from the induction hypothesis.

□

B ADDITIONAL MATERIAL IN SUPPORT OF SECTION 5

This section contains: a definition for an algorithmic version of the constrained type system rules (where the sub-type rules have been distributed throughout the other rules in the standard way), the proof of its correctness in the form of a soundness and completeness proof (relative to the original constraint type system rules), a type inference algorithm, and its correctness in the form of a soundness and completeness proof (relative to the algorithmic typing rules).

Definition B.1. Given a set of typing and subtype formulas Γ, write $\mathsf{Con}(\Gamma)$ for the subset of subtype formulas, and $\mathsf{Typ}(\Gamma)$ for the typing formulas. Given two sets of typing and subtype formulas Γ and Γ' write $\Gamma \sqsubseteq \Gamma'$ just if for all typings $M : A \in \Gamma$, there is a typing $M : B \in \Gamma'$ and $\Gamma \vdash A \sqsubseteq B$.

Theorem B.2 (Soundness of Algorithmic Type Assignment). *For all* Γ, Δ, *if* $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$ *then* $\Gamma \cup C \vdash \Delta$.

Proof. We proceed by induction on the derivation of $\vdash_2$.

Case(Inst2):

This case is trivial by the use of (GVar).

Case(Var2):

Let $\Gamma, x : A$ be the context with $A \sqsubseteq B$.

By (LVar), we have $\Gamma, x : A \vdash x : A$.

By (SubR), we have $\Gamma, x : A \vdash x : B$ as required.

Case(AbsR2):

Let $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 (\lambda x.\, M) : B_1 \rightarrow A_1$ and $\mathsf{Typ}(\Gamma), x : B_2 \mid \mathsf{Con}(\Gamma) \vdash_2 M : A_2$ with $B_1 \sqsubseteq B_2$ and $A_2 \sqsubseteq A_1$.

By the inductive hypothesis, $\Gamma, x : B_2 \vdash M : A_2$.

By (SubR) we have $\Gamma, x : B_2 \vdash M : A_1$.

By (SubL) we have $\Gamma, x : B_1 \vdash M : A_1$.

By (AbsR) we have $\Gamma \vdash (\lambda x.\, M) : B_1 \rightarrow A_1$ as required.

Case(AbnR2):

Structural

$$\text{(Inst2)}\ \frac{}{\Gamma,\ f : \forall\, \vec{a}\,.C' \Rightarrow A \mid C \vdash_2 f : A'}\ C \vdash C'[\vec{B}/\vec{a}] \text{ and } C \vdash A[\vec{B}/\vec{a}] \sqsubseteq A'$$

$$\text{(Var2)}\ \frac{}{\Gamma,\ x : A \mid C \vdash_2 x : B}\ C \vdash A \sqsubseteq B \qquad \text{(VarK2)}\ \frac{}{\Gamma \mid C \vdash_2 x : A}\ C \vdash \mathsf{Ok} \sqsubseteq A$$

Functions

$$\text{(AbsR2)}\ \frac{\Gamma,\ x : B_1 \mid C \vdash_2 M : B_2}{\Gamma \mid C \vdash_2 (\lambda x.\, M) : A}\ C \vdash B_1 \to B_2 \sqsubseteq A \qquad \text{(AbnR2)}\ \frac{\Gamma,\ M : B_1 \mid C \vdash_2 x : B_2}{\Gamma \mid C \vdash_2 (\lambda x.\, M) : A}\ C \vdash B_2 \rightarrowtail B_1 \sqsubseteq A$$

$$\text{(AppR2)}\ \frac{\Gamma \mid C \vdash_2 M : B_1 \qquad \Gamma \mid C \vdash_2 N : B_2}{\Gamma \vdash_2 (M\ N) : A}\ C \vdash B_1 \sqsubseteq B_2 \to A$$

$$\text{(AppL2)}\ \frac{\Gamma \mid C \vdash_2 M : B_1 \qquad \Gamma,\ N : B_2 \vdash_2 \Delta}{\Gamma,\ (M\ N) : A \mid C \vdash_2 \Delta}\ C \vdash B_1 \sqsubseteq B_2 \rightarrowtail A$$

Constructors

$$\text{(CnsL2)}\ \frac{\Gamma,\ M_i : A_i \mid C \vdash_2 \Delta}{\Gamma,\ c(M_1, \ldots, M_n) : A \mid C \vdash_2 \Delta}\ C \vdash A \sqsubseteq \Sigma_{d \in C \setminus \{c\}}\, d(\overrightarrow{A_d}) + c(A_1, \ldots, A_n)$$

$$\text{(CnsR2)}\ \frac{\Gamma \mid C \vdash_2 M_i : A_i\ (\forall i)}{\Gamma \mid C \vdash_2 c(M_1, \ldots, M_n) : A}\ C \vdash c(A_1, \ldots, A_n) \sqsubseteq A$$

Pattern Matching

$$\text{(MchL2)}\ \frac{\begin{array}{c}\Gamma,\ (M, P_i) : A'_i \mid C \vdash_2 \Delta\ (\forall i) \\ \Gamma,\ P_i : A_x \mid C \vdash_2 x : B_x\ (\forall i. \forall x \in \mathsf{FV}(p_i))\end{array}}{\Gamma,\ \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\} : A \mid C \vdash_2 \Delta}\ \begin{array}{l} C \vdash A \sqsubseteq A_i\ (\forall i) \\ C \vdash (B_i, A_i) \sqsubseteq A'_i\ (\forall i) \\ C \vdash A \sqsubseteq A_x\ (\forall i. \forall x \in \mathsf{FV}(p_i)) \\ C \vdash p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq B_i\ (\forall i)\end{array}$$

$$\text{(MchR2)}\ \frac{\begin{array}{c}\Gamma \mid C \vdash_2 M : B \\ \Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \mid C \vdash_2 P_i : A_i\ (\forall i)\end{array}}{\Gamma \mid C \vdash_2 \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\} : A}\ \begin{array}{l} C \vdash B \sqsubseteq \Sigma_{i=1}^{k} p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \\ C \vdash A_i \sqsubseteq A\ (\forall i)\end{array}$$

Fixpoints and Evaluation

$$\text{(FixR2)}\ \frac{\Gamma,\ f : A \mid C \vdash_2 M : B}{\Gamma \mid C \vdash_2 \mathsf{fix}\, f.\, M : A}\ C \vdash B \sqsubseteq A$$

$$\text{(CnsK2)}\ \frac{\Gamma,\ M_i : B \mid C \vdash_2 \Delta}{\Gamma,\ c(M_1, \ldots, M_n) : A \mid C \vdash_2 \Delta}\ C \vdash \mathsf{Ok} \sqsubseteq B \qquad \text{(FunK2)}\ \frac{\Gamma,\ M : B \mid C \vdash_2 \Delta}{\Gamma,\ M\ N : A \mid C \vdash_2 \Delta}\ C \vdash \mathsf{Ok} \rightarrowtail A \sqsubseteq B$$

Fig. 6. Algorithmic constrained type assignment

Disjointness

$$\text{(CnsDL21)}\ \frac{}{\Gamma,\ c(M_1,\ldots,M_n) : A \mid C \vdash \Delta}\ CA \sqsubseteq B_1 \rightarrow B_2$$

$$\text{(CnsDL22)}\ \frac{}{\Gamma,\ c(M_1,\ldots,M_n) : A \mid C \vdash \Delta}\ C \vdash A \sqsubseteq B_1 \succ B_2$$

$$\text{(CnsDL23)}\ \frac{}{\Gamma,\ c(M_1,\ldots,M_n) : A \mid C \vdash \Delta}\ C \vdash A \sqsubseteq \Sigma_{d \in C \setminus \{c\}}\ d(\overrightarrow{A_d})$$

$$\text{(AbsDL2)}\ \frac{}{\Gamma,\ \lambda x.\, M : A \mid C \vdash \Delta}\ C \vdash A \sqsubseteq \Sigma_{c \in C} c(\overrightarrow{A_c})$$

Fig. 7. Algorithmic constrained type assignment continued

Let $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 (\lambda x.\, M) : B_1 \succ A_1$ and $\mathsf{Typ}(\Gamma),\ M : A_2 \mid \mathsf{Con}(\Gamma) \vdash_2 x : B_2$ with $B_2 \sqsubseteq B_1$ and $A_1 \sqsubseteq A_2$.
By the inductive hypothesis, $\Gamma,\ M : A_2 \vdash x : B_2$.
By (SubR) we have $\Gamma,\ M : A_2 \vdash x : B_1$.
By (SubL) we have $\Gamma,\ M : A_1 \vdash x : B_1$.
By (AbnR) we have $\Gamma \vdash_2 (\lambda x.\, M) : B_1 \succ A_1$ as required.

Case(AppR2):
Let $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 (M\ N) : \tau_3,\ \mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 M : \tau_1$, and $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 N : \tau_2$ with $\tau_1 \sqsubseteq \tau_2 \rightarrow \tau_3$.
By the inductive hypothesis, we have $\Gamma \vdash M : \tau_1$ and $\Gamma \vdash N : \tau_2$. By (SubR) on M, we have $\Gamma \vdash M : \tau_2 \rightarrow \tau_3$.
By (AppR) we have $\Gamma \vdash (M\ N) : \tau_3$ as required.

Case(AppL2):
Let $\mathsf{Typ}(\Gamma),\ (M\ N) : \tau_3 \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta,\ \mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 M : \tau_1$, and $\mathsf{Typ}(\Gamma),\ N : \tau_2 \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$ with $\tau_1 \sqsubseteq \tau_2 \succ \tau_3$.
By the inductive hypothesis, we have $\Gamma \vdash M : \tau_1$ and $\Gamma,\ N : \tau \vdash_2 \Delta$.
By (SubR) on M we have $\Gamma \vdash M : \tau_2 \succ \tau_3$.
By (AppL), we have $\Gamma,\ (M\ N) : \tau_3 \vdash \Delta$ as required.

Case(ConsL2):
Let $\mathsf{Typ}(\Gamma),\ c(M_1,\ldots,M_n) : \tau \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$ and $\mathsf{Typ}(\Gamma),\ M_i : A_i \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$ with $\tau \sqsubseteq \Sigma_{d \in C \setminus \{c\}}(d(\vec{A_d})) + c(A_1,\ldots,A_n)$.
By the inductive hypothesis, $\Gamma,\ M_i : A_i \vdash \Delta$.
By (CnsL), we have $\Gamma,\ c(M_1,\ldots,M_n) : c(A_1,\ldots,A_n) + \Sigma_{d \in C \setminus \{c\}}(d(\vec{A_d})) \vdash \Delta$.
By (SubL), we have $\Gamma,\ c(M_1,\ldots,M_n) : \tau \vdash \Delta$ as required.

Case(ConsR2):
Let $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 c(_1,\ldots,M_n) : \tau$ and $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 M_i : A_i\ (\forall i)$ with $c(A_1,\ldots,A_n) \sqsubseteq \tau$.
By the inductive hypothesis, we have $\Gamma \mid C \vdash M_i : A_i\ (\forall i)$.
By (CnsR), we have $\Gamma \vdash_2 c(_1,\ldots,M_n) : c(A_1,\ldots,A_n)$.
By (SubR), we have $\Gamma \vdash c(_1,\ldots,M_n) : \tau$ as required.

Case(MchL2):

Let $\mathsf{Typ}(\Gamma)$, $\mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k}(p_i \mapsto P_i)\} : A \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$, $\mathsf{Typ}(\Gamma)$, $(M, P_i) : A_i' \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta\ (\forall i)$ and $\mathsf{Typ}(\Gamma)$, $P_i : A_x \mid \mathsf{Con}(\Gamma) \vdash_2 x : B_x\ (\forall i.\forall x \in \mathsf{FV} p_i)$ where $A \sqsubseteq A_x\ (\forall i.\forall x \in \mathsf{FV}(p_i))$, $A \sqsubseteq A_i\ (\forall i.)$, $(B_i, A_i) \sqsubseteq A_i'$, and $p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq B_i\ (\forall i.)$.

By the inductive hypothesis, Γ, $(M, P_i) : A_i' \vdash \Delta\ (\forall i)$ and Γ, $P_i : A_x \vdash x : B_x\ (\forall i.\forall x \in \mathsf{FV} p_i)$.

By (SubL) (applied to (M, P_i) twice and P_i once), we have Γ, $(M, P_i) : (p_i[B_x/x \mid x \in \mathsf{FV}(p_i)], A) \vdash \Delta\ (\forall i)$ and Γ, $P_i : A \vdash x : B_x\ (\forall i.\forall x \in \mathsf{FV} p_i)$.

By (MchL), we have Γ, $\mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k}(p_i \mapsto P_i)\} : A \vdash \Delta$ as required.

Case(MchR2):

Let $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k}(p_i \mapsto P_i)\} : A$, $\mathsf{Typ}(\Gamma) \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \mid \mathsf{Con}(\Gamma) \vdash_2 P_i : A_i\ (\forall i)$, and $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 M : B$ where $B \sqsubseteq \Sigma_{i=1}^{k} p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$ and $A_i \sqsubseteq A\ (\forall i)$.

By the inductive hypothesis we have $\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A_i\ (\forall i)$ and $\Gamma \vdash M : B$.

By (SubR) (applied to M and P_i), we have $\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A\ (\forall i)$ and $\Gamma \vdash M : p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$.

By (MchR), we have $\Gamma \vdash \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k}(p_i \mapsto P_i)\} : A$.

Case(FixR2):

Let $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 \mathsf{fix}\, f.M : A$, $\mathsf{Typ}(\Gamma)$, $f : A \mid \mathsf{Con}(\Gamma) \vdash_2 M : B$, and $B \sqsubseteq A$.

By the inductive hypothesis, we have Γ, $f : A \vdash M : B$.

By (SubR), we have Γ, $f : A \vdash M : A$.

By (FixR), we have $\Gamma \vdash \mathsf{fix}\, f.M : A$ as required.

Case(VarK2):

This case is trivial by the use of (VarK).

Case(CnsK2):

This case is trivial by the use of (CnsK).

Case(FunK2):

Let $\mathsf{Typ}(\Gamma)$, $M\ N : A \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$, $\mathsf{Typ}(\Gamma)$, $M : B \vdash_2 \Delta$, and $\mathsf{Ok} \rightarrowtail A \sqsubseteq B$.

By the inductive hypothesis, we have Γ, $M : B \vdash \Delta$.

By (SubL), we have Γ, $M : \mathsf{Ok} \rightarrowtail A \vdash \Delta$.

By (FunK), we have Γ, $M\ N : \mathsf{Ok} \vdash \Delta$.

By (SubL), we have Γ, $M\ N : A \vdash \Delta$ as required.

Case(CnsDL21):

This case is trivial by the use of (CnsDL) where the type A is an arrow.

Case(CnsDL22):

This case is trivial by the use of (CnsDL) where the type A is an arrow.

Case(CnsDL23):

This case is trivial by the use of (CnsDL) where the type A is a sum of constructor types that does not include c.

Case(AbsDL2):

This case is trivial by the use of (AbsDL) where the type A is a sum of constructor types.

□

Theorem B.3 (Completeness of Algorithmic Type Assignment). *For all* $\Gamma, \Gamma', \Delta, \Delta'$, *if the following hold:*

(1) $\Gamma \vdash \Delta$

(2) $\Delta \sqsubseteq \Delta'$

(3) $\Gamma' \sqsubseteq \Gamma$

Then $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 \Delta'$.

Proof. We proceed by induction on the derivation of $\vdash$.

Case(GVar):

This case is trivial by the use of (Inst2).

Case(LVar):

Let $\Gamma, x : A \vdash x : A,\ \Gamma', x : C \sqsubseteq \Gamma, x : A$, and $A \sqsubseteq B$.

Then by (Var2), $\mathsf{Typ}(\Gamma'), x : C \mid \mathsf{Con}(\Gamma), C \sqsubseteq B \vdash_2 x : B$.

Case(SubL):

Let $\Gamma, M : A \vdash \Delta,\ \Gamma, M : B \vdash \Delta$, and $A \sqsubseteq B$.

By the inductive hypothesis, for all Γ', Δ' such that $\Gamma' \sqsubseteq \Gamma$, $M : A$ and $\Delta \sqsubseteq \Delta'$ we have $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 \Delta'$.

As $A \sqsubseteq B$, $M : A$ is an included entry in an instance of Γ'.

Thus, for all $\Gamma'' \sqsubseteq \Gamma$, $A' \sqsubseteq A$, and $\Delta \sqsubseteq \Delta''$, we have $\mathsf{Typ}(\Gamma''), M : A' \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$.

Case(SubR):

Let $\Gamma \vdash M : A,\ \Gamma \vdash M : B$, and $B \sqsubseteq A$.

By the inductive hypothesis, for all Γ', B' such that $\Gamma' \sqsubseteq \Gamma$ and $B \sqsubseteq B'$ we have $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 M : B'$.

Thus, as $B \sqsubseteq A$ we have for all A' such that $A \sqsubseteq A'$, $B \sqsubseteq A'$ and so $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 M : A'$ as required.

Case(AbsR):

Let $\Gamma \vdash (\lambda x.\, M) : B \to A$ and $\Gamma, x : B \vdash M : A$.

By the inductive hypothesis, for all Γ', B', A' such that$\Gamma' \sqsubseteq \Gamma$, $B' \sqsubseteq B$, and $A \sqsubseteq A'$, we have $\mathsf{Typ}(\Gamma'), x : B' \mid \mathsf{Con}(\Gamma') \vdash_2 M : A'$.

Let $\Gamma'' \sqsubseteq \Gamma$, $B' \sqsubseteq B$, and $A \sqsubseteq A'$. Then $B \to A \sqsubseteq B' \to A'$ and $\mathsf{Typ}(\Gamma''), x : B' \mid \mathsf{Con}(\Gamma'') \vdash_2 M : A'$.

Then, by (AbsR2), we have $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 \lambda x.\, M : B' \to A'$ as required.

Case(AbnR):

Let $\Gamma \vdash (\lambda x.\, M) : B \rightarrowtail A$ and $\Gamma, M : A \vdash x : B$.

By the inductive hypothesis, for all Γ', A', B' such that $\Gamma' \sqsubseteq \Gamma$, $B \sqsubseteq B'$ and $A' \sqsubseteq A$, we have $\mathsf{Typ}(\Gamma'), M : A' \mid \mathsf{Con}(\Gamma') \vdash_2 x : B'$.

Let $\Gamma'' \sqsubseteq \Gamma$, $A' \sqsubseteq A$, $B \sqsubseteq B'$. Then $B \rightarrowtail A \sqsubseteq B' \rightarrowtail A'$ and $\mathsf{Typ}(\Gamma''), M : A' \mid \mathsf{Con}(\Gamma'') \vdash_2 x : B'$.

Then, by (AbnR2), we have $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 (\lambda x.\, M) : B' \rightarrowtail A'$ as required.

Case(AppL):

Let $\Gamma, (M\ N) : A \vdash \Delta,\ \Gamma \vdash M : B \rightarrowtail A$, and $\Gamma, N : B \vdash \Delta$.

By the inductive hypothesis, for all Γ', Δ', A', B' such that $\Gamma' \sqsubseteq \Gamma$, $\Delta \sqsubseteq \Delta'$, $B \rightarrowtail A \sqsubseteq B' \rightarrowtail A'$, and $B \sqsubseteq B'$ we have $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 M : B' \rightarrowtail A'$ and $\mathsf{Typ}(\Gamma'), N : B' \mid \mathsf{Con}(\Gamma') \vdash_2 \Delta'$.

Let $\Gamma'' \sqsubseteq \Gamma$, $A' \sqsubseteq A$, and $\Delta \sqsubseteq \Delta'$. Then, for all B' such that $B \rightarrowtail A \sqsubseteq B' \rightarrowtail A'$ and $B \sqsubseteq B'$, we have $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 M : B' \rightarrowtail A'$ and $\mathsf{Typ}(\Gamma''), N : B' \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$.

Then, by (AppL2), as the side condition is trivially satisfied, we have $\mathsf{Typ}(\Gamma''), (M\ N) : A' \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$.

Case(AppR):

Let $\Gamma \vdash (M\ N) : A,\ \Gamma \vdash M : B \to A$, and $\Gamma \vdash N : B$.

By the inductive hypothesis, for all Γ', A', B' we have $\Gamma' \sqsubseteq \Gamma$, $B \sqsubseteq B'$, $B \to A \sqsubseteq B' \to A'$, $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 M : B' \to A'$, and $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 N : B'$.

Let $\Gamma'' \sqsubseteq \Gamma$ and $A \sqsubseteq A'$. Then, for all B' such that $B \sqsubseteq B'$ and $B \to A \sqsubseteq B' \to A'$, we have $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 M : B' \to A'$, and $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 N : B'$.

Then, by (AppR2), as the side condition is trivially satisfied, we have $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 (M\ N) : A'$ as required.

Case(CnsL):

Let $\Gamma,\ c(M_1,\ldots,M_n) : c(A_1,\ldots,A_n) + \Sigma_{d\in C\setminus\{c\}}(d(\vec{A_d})) \vdash \Delta$ and $\Gamma,\ M_i : A_i \vdash \Delta$ for some i.
By the inductive hypothesis, for all Γ', A_i', Δ' we have $\Gamma' \sqsubseteq \Gamma,\ \Delta \sqsubseteq \Delta',\ A_i' \sqsubseteq A_i$, and $\mathsf{Typ}(\Gamma'),\ M_i : A_i' \mid \mathsf{Con}(\Gamma') \vdash_2 \Delta'$.
Let $\Gamma'' \sqsubseteq \Gamma,\ \Delta \sqsubseteq \Delta''$ and $A_j' \sqsubseteq A_j (\forall j)$. Then $c(A_1',\ldots,A_n') + \Sigma_{d\in C\setminus\{c\}}(d(\vec{A_d'})) \sqsubseteq c(A_1,\ldots,A_n) + \Sigma_{d\in C\setminus\{c\}}(d(\vec{A_d}))$ and $\mathsf{Typ}(\Gamma''),\ M_i : A_i' \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$.
Then, by (ConsL2), we have $\mathsf{Typ}(\Gamma''),\ c(M_1,\ldots,M_n) : c(A_i',\ldots,A_n') + \Sigma_{d\in C\setminus\{c\}}(d(\vec{A_d'})) \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$.

Case(CnsR):
Let $\Gamma \vdash c(M_1,\ldots,M_n) : c(A_1,\ldots,A_n)$ and $\Gamma \vdash M_i : A_i$ for all i.
By the inductive hypothesis, for all Γ', A_i' we have $\Gamma' \sqsubseteq \Gamma,\ A_i \sqsubseteq A_i'$, and $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 M_i : A_i'$ for all i.
Let $\Gamma'' \sqsubseteq \Gamma$ and $A_i \sqsubseteq A_i'\ (\forall i)$. Then $c(A_i,\ldots,A_n) \sqsubseteq c(A_i',\ldots,A_n')$ and $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 M_i : A_i'\ (\forall i)$.
Then, by (ConsR2), we have $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 c(M_i,\ldots,M_n) : c(A_i',\ldots,A_n')$ as required.

Case(MchL):
Let $\Gamma,\ \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^k (p_i \mapsto P_i)\} : A \vdash \Delta,\ \Gamma, P_i : A \vdash x : B_x\ (\forall i.\forall x \in \mathsf{FV}(p_i))$, and $\Gamma,\ (M, P_i) : (p_i[B_x/x \mid x \in \mathsf{FV}(p_i)], A) \vdash \Delta\ (\forall i)$.
By the inductive hypothesis we have:

(1) $\forall \Gamma', A_x, B_x'.\Gamma' \sqsubseteq \Gamma \wedge A_x \sqsubseteq A \wedge B_x \sqsubseteq B_x' \Rightarrow \mathsf{Typ}(\Gamma'),\ P_i : A_x \mid \mathsf{Con}(\Gamma') \vdash_2 x : B_x'$
(2) $\forall \Gamma', \tau, \Delta'.\Gamma' \sqsubseteq \Gamma \wedge \tau \sqsubseteq (p_i[B_x/x \mid x \in \mathsf{FV}(p_i)], A) \wedge \Delta \sqsubseteq \Delta' \Rightarrow \mathsf{Typ}(\Gamma'),\ (M, P_i) : (p_i[B_x'/x \mid x \in \mathsf{FV}(p_i)], A') \mid \mathsf{Con}(\Gamma') \vdash_2 \Delta'$

Let $\Gamma'' \sqsubseteq \Gamma,\ A'' \sqsubseteq A$, and $\Delta \sqsubseteq \Delta''$.
By instantiating (1): Γ' with Γ'', A_x with A'', and B_x' with B_x gives $\mathsf{Typ}(\Gamma''),\ P_i : A'' \mid \mathsf{Con}(\Gamma'') \vdash_2 x : B_x$.
By instantiating (2): Γ' with Γ'', τ with (B_x, A''), and Δ' with Δ'' gives $\mathsf{Typ}(\Gamma''),\ (M, P_i) : (p_i[B_x/x \mid x \in \mathsf{FV}(p_i)], A'') \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$.
Finally, by (MchL2), as the side conditions are trivially satisfied, we have $\mathsf{Typ}(\Gamma''),\ \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^k (p_i \mapsto P_i)\} : A'' \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$ as required.

Case(MchR):
Let $\Gamma \mid C \vdash \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^k (p_i \mapsto P_i)\} : A,\ \Gamma \vdash M : \Sigma_{i=1}^k p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$, and $\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A (\forall i)$.
By the inductive hypothesis, for all Γ', A', B, B_x' such that $\Gamma' \sqsubseteq \Gamma,\ \Sigma_{i=1}^k p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq B, (\forall i) B_x' \sqsubseteq B_x$, and $A \sqsubseteq A'$, we have

(1) $\mathsf{Typ}(\Gamma') \mid \mathsf{Con}(\Gamma') \vdash_2 M : B$
(2) $\mathsf{Typ}(\Gamma') \cup \{x : B_x' \mid x \in \mathsf{FV}(p_i)\} \mid \mathsf{Con}(\Gamma') \vdash_2 P_i : A' (\forall i)$

Let $\Gamma'' \sqsubseteq \Gamma$ and $A \sqsubseteq A''$. We want to show $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^k (p_i \mapsto P_i)\} : A''$.
Instantiating (1): Γ' with Γ'' and B with $\Sigma_{i=1}^k p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$ gives $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 M : \Sigma_{i=1}^k p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$.
Instantiating (2): Γ' with Γ'', B_x' with B_x, and A' with A'', gives $\mathsf{Typ}(\Gamma'') \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \mid \mathsf{Con}(\Gamma'') \vdash_2 P_i : A'' (\forall i)$.
Finally, by (MchR2), we have $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^k (p_i \mapsto P_i)\} : A''$ as required.

Case(FixR):

Let $\Gamma \vdash \text{fix}\, f.\, M : A$ and $\Gamma,\ f : A \vdash M : A$.
By the inductive hypothesis, for all Γ', B, C such that $\Gamma' \sqsubseteq \Gamma$, $B \sqsubseteq A$, and $A \sqsubseteq C$, we have $\mathsf{Typ}(\Gamma'),\ f : B \mid \mathsf{Con}(\Gamma') \vdash_2 M : C$.
Let $\Gamma'' \sqsubseteq \Gamma$ and $A \sqsubseteq A''$.
Instantiating Γ' with Γ'', and B with A'' gives $\mathsf{Typ}(\Gamma''),\ f : A'' \mid \mathsf{Con}(\Gamma'') \vdash_2 M : C$.
By (FixR2), we have $\mathsf{Typ}(\Gamma'') \mid \mathsf{Con}(\Gamma'') \vdash_2 \text{fix}\, f.\, M : A''$ as required.

Case(VarK):
This case is trivial by the use of (VarK2).

Case(CnsK):
This case is trivial by the use of (CnsK2) with A as Ok.

Case(FunK):
Let $\Gamma,\ M\ N : \mathsf{Ok} \vdash \Delta$ and $\Gamma,\ M : \mathsf{Ok} \rightarrowtail A \vdash \Delta$.
By the inductive hypothesis, for all Γ', B, Δ' such that $\Gamma' \sqsubseteq \Gamma'$, $B \sqsubseteq \mathsf{Ok} \rightarrowtail A$, and $\Delta \sqsubseteq \Delta'$, we have $\mathsf{Typ}(\Gamma'),\ M : B \mid \mathsf{Con}(\Gamma') \vdash_2 \Delta'$.
Let $\Gamma'' \sqsubseteq \Gamma$ and $\Delta' \sqsubseteq \Delta''$.
By instantiating Γ' with Γ'', Δ' with Δ'', and B with $\mathsf{Ok} \rightarrowtail \mathsf{Ok}$, we have $\mathsf{Typ}(\Gamma''),\ M : \mathsf{Ok} \rightarrowtail \mathsf{Ok} \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$.
By (FunK2), we have $\mathsf{Typ}(\Gamma''),\ M\ N : \mathsf{Ok} \mid \mathsf{Con}(\Gamma'') \vdash_2 \Delta''$ as required.

Case(CnsDL):
Let $\Gamma,\ c(M_1, \ldots, M_n) : A \vdash \Delta$ where A is and arrow type or of the shape $\Sigma_{d \in I}$ with $c \notin I$.
Let $\Gamma' \sqsubseteq \Gamma$, $A' \sqsubseteq A$, and $\Delta \sqsubseteq \Delta'$.
If A is a sum of constructor types that does not include c, then so is A'. Thus, by (CnsDL23), we have $\Gamma',\ c(M_1, \ldots, M_n) : A' \vdash \Delta'$.
If A is a sufficiency arrow $(\tau_1 \rightarrow \tau_2)$, then so is A'. Thus, by (CnsDL21), we have $\Gamma',\ c(M_1, \ldots, M_n) : A' \vdash \Delta'$.
If A is a necessity arrow $(\tau_1 \rightarrowtail \tau_2)$, then so is A'. Thus, by (CnsDL22), we have $\Gamma',\ c(M_1, \ldots, M_n) : A' \vdash \Delta'$.

□

B.1 Inference Algorithm

This (sub-)section contains a definition for the constrained type system inference algorithm and a proof of its correctness in the form of a soundness and correctness proof (relative to the algorithmic typing rules).

```
InferR(Γ, M) = case M of
Var x ⟶ { (Γ | {a ⊑ b} ⊢ M : b) | b = freshVar,
        a = { t    (x : t) ∈ Γ
            { Ok   otherwise
  } ∪ { (Γ | C[b̄/ā] ∪ {a[b̄/ā] ⊑ a'} ⊢ M : a') | b̄ = freshVar, a' = freshVar,
        (x : ∀ ā. C ⇒ a) ∈ Γ
  }
App P Q ⟶ { (Γ | c1 ∪ c2 ∪ {b ⊑ c ⟶ a} ⊢ M : a) | a = freshVar,
        (Γ | c1 ⊢ P : b) ∈ inferR(Γ, P),
        (Γ | c2 ⊢ Q : c) ∈ inferR(Γ, Q)
  }
Abs x N ⟶ { (Γ | c ∪ {t ⟶ b ⊑ a} ⊢ M : a) | a = freshVar, t = freshVar,
        (Γ ∪ {x : t} | c ⊢ N : b) ∈ inferR(Γ ∪ {x : t}, N)
  } ∪ { (Γ | c ∪ {t ⟞ b ⊑ a} ⊢ M : a) | a = freshVar, t = freshVar,
```

$$
\begin{aligned}
&\quad (\Gamma \cup \{\mathsf{N} : \mathsf{b}\} \mid \mathsf{c} \vdash \mathsf{x} : \mathsf{t}) \in \mathsf{inferL}(\Gamma, \mathsf{N}, \{\mathsf{x} : \mathsf{t}\}) \\
&\} \\
&\mathsf{Cons\ c\ ms} \longrightarrow \{\, (\Gamma \mid \textstyle\bigcup_{i=1}^{n}(C_i) \cup \{\mathsf{c}(a_1, \ldots, a_n) \sqsubseteq \mathsf{a}\} \vdash \mathsf{M} : \mathsf{a}) \mid \mathsf{a} = \mathsf{freshVar}, \\
&\quad (\Gamma \mid C_i \vdash m_i : a_i)_{i=1}^{n} \in \Pi_{m \in \mathsf{ms}}(\mathsf{inferR}(\Gamma, \mathsf{m})) \\
&\} \cup \{\, \Gamma \mid \{\mathsf{c} \sqsubseteq \mathsf{a}\} \vdash \mathsf{M} : \mathsf{a} \mid \mathsf{a} = \mathsf{freshVar}, \mathsf{ms} = \{\,\}\,\} \\
&\mathsf{Match\ Q}\,\{|_{i=1}^{k}\ \mathsf{p}_i \longrightarrow \mathsf{P}_i\} \longrightarrow \{\, (\Gamma \mid \textstyle\bigcup_{i=1}^{k}(C_i \cup \{b \sqsubseteq p_i[b_x/x \mid x \in \mathsf{FV}(p_i)], a_i \sqsubseteq a\}) \cup \mathsf{C} \vdash \mathsf{M} : \mathsf{a}) \mid \\
&\quad \mathsf{a} = \mathsf{freshVar}, \overline{b_x} = \overline{\mathsf{freshVar}}, \\
&\quad (\Gamma \mid \mathsf{C} \vdash \mathsf{Q} : \mathsf{b}) \in \mathsf{inferR}(\Gamma, \mathsf{Q}), \\
&\quad (\Gamma \cup \{\overline{(x : b_x)}\} \mid C_i \vdash P_i : a_i)_{i=1}^{k} \in \Pi_{i=1}^{k}(\mathsf{inferR}(\Gamma \cup \{\overline{(x : b_x)}\}, P_i)) \\
&\} \\
&\mathsf{Fix\ f\ m} \longrightarrow \{\, (\Gamma \mid \mathsf{c} \cup \{\mathsf{b} \sqsubseteq \mathsf{a}\} \vdash \mathsf{M} : \mathsf{a}) \mid \mathsf{a} = \mathsf{freshVar}, \\
&\quad (\Gamma \cup \{\mathsf{f} : \mathsf{a}\} \mid \mathsf{c} \vdash \mathsf{m} : \mathsf{b}) \in \mathsf{inferR}(\Gamma \cup \{\mathsf{f} : \mathsf{a}\}, \mathsf{m}) \\
&\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{InferL}(\Gamma, M, \Delta) = \{\, \Gamma \cup \{M : a\} \mid \{\mathsf{Ok} \sqsubseteq b\} \vdash \Delta \mid a = \mathsf{freshVar}, (x : b) \in d \,\} \cup \textbf{case } M \textbf{ of} \\
&\quad \mathsf{Var}\ x \to \{\, \Gamma \cup \{M : a\} \mid \{a \sqsubseteq b\} \vdash \Delta \mid a = \mathsf{freshVar}, (x : b) \in \Delta \,\} \\
&\quad \mathsf{App}\ P\ Q \to \{\, (\Gamma \cup \{M : a\} \mid c1 \cup c2 \cup \{b \sqsubseteq c \rightarrowtail a\} \vdash d) \mid a = \mathsf{freshVar}, \\
&\qquad (\Gamma \mid c1 \vdash P : b) \in \mathsf{inferR}(\Gamma, P), \\
&\qquad (\Gamma \cup \{Q : c\} \mid c2 \vdash \Delta) \in \mathsf{inferL}(\Gamma, Q, \Delta) \\
&\quad \} \cup \{\, (\Gamma \cup \{M : a\} \mid c \cup \{\mathsf{Ok} \rightarrowtail a \sqsubseteq b\} \vdash \Delta) \mid a = \mathsf{freshVar}, \\
&\qquad (\Gamma \cup \{P : b\} \mid c \vdash \Delta) \in \mathsf{inferL}(\Gamma, P, \Delta) \\
&\quad \} \\
&\quad \mathsf{Abs}\ x\ N \to \{\, (\Gamma \cup \{M : a\} \mid \{a \sqsubseteq \Sigma_{c \in C}(c(\overline{a}))\} \vdash \Delta) \mid a = \mathsf{freshVar}, \overline{a} = \overline{\mathsf{freshVar}} \,\} \\
&\quad \mathsf{Cons}\ \kappa\ ms \to \{\, (\Gamma \cup \{M : a\} \mid c \cup \{a \sqsubseteq \Sigma_{\kappa' \in C \setminus \{\kappa\}}(\kappa'(\overline{a_{\kappa'}})) + \kappa(a_1, \ldots, b, \ldots, a_n)\} \vdash \Delta) \mid \\
&\qquad a = \mathsf{freshVar}, \overline{a_{\kappa'}} = \overline{\mathsf{freshVar}}, (a)_{i=1}^{n} = \overline{\mathsf{freshVar}}, \\
&\qquad m_i \in ms, \\
&\qquad (\Gamma \cup \{m_i : b\} \mid c \vdash \Delta) \in \mathsf{inferL}(\Gamma, m, \Delta) \\
&\quad \} \cup \{\, (\Gamma \cup \{M : a\} \mid c \cup \{\mathsf{Ok} \sqsubseteq b\} \vdash \Delta) \mid a = \mathsf{freshVar}, \\
&\qquad m_i \in ms, \\
&\qquad (\Gamma \cup \{m_i : b\} \mid c \vdash d) \in \mathsf{inferL}(\Gamma, m, \Delta) \\
&\quad \} \cup \{\, (\Gamma \cup \{M : a\} \mid \{a \sqsubseteq b_1 \to b_2\} \vdash \Delta) \mid a = \mathsf{freshVar}, b_1 = \mathsf{freshVar}, b_2 = \mathsf{freshVar} \,\} \\
&\quad \cup \{\, (\Gamma \cup \{M : a\} \mid \{a \sqsubseteq b_1 \rightarrowtail b_2\} \vdash \Delta) \mid a = \mathsf{freshVar}, b_1 = \mathsf{freshVar}, b_2 = \mathsf{freshVar} \,\} \\
&\quad \cup \{\, (\Gamma \cup \{M : a\} \mid \{a \sqsubseteq \Sigma_{c \in C \setminus \kappa}(c(\overline{a}))\} \vdash \Delta) \mid a = \mathsf{freshVar}, \overline{a} = \overline{\mathsf{freshVar}} \,\} \\
&\quad \mathsf{Match}\ q\ \{|_{i=1}^{k}\ p_i \to P_i\} \to \{\, (\Gamma \cup \{M : a\} \mid c \vdash \Delta) \mid \\
&\qquad a = \mathsf{freshVar}, \ (a_i, b_i)_{i=1}^{k} = \overline{\mathsf{freshVar}}, \ (b_{(i,x)})_{(i,x) \in [1..k] \times \mathsf{FV}(p_i)} = \overline{\mathsf{freshVar}}, \\
&\qquad (\Gamma \cup \{(q, P_i) : a'_i\} \mid c_i \vdash \Delta)_{i=1}^{k} \in \Pi_{i=1}^{k}(\mathsf{inferL}(\Gamma, (q, P_i), \Delta)), \\
&\qquad (\Gamma \cup \{P_i : a_{(i,x)}\} \mid c'_{(i,x)} \vdash \Delta)_{(i,x)} \in \Pi_{(i,x) \in [1..k] \times \mathsf{FV}(p_i)}(\mathsf{inferL}(\Gamma, P_i, \{x : b_{(i,x)}\})), \\
&\qquad c = \bigcup_{i=1}^{k}(c_i \cup \{a \sqsubseteq a_i, (a_i, b_i) \sqsubseteq a'_i, p_i[b_{(i,x)}/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq b_i\}) \cup \\
&\qquad\quad \bigcup_{(i,x) \in [1..k] \times \mathsf{FV}(p_i)}(c'_{(i,x)} \cup \{a \sqsubseteq a_{(i,x)}\}) \\
&\quad \} \\
&\quad \mathsf{Fix}\ f\ n \to \{\,\}
\end{aligned}
$$

Theorem B.4 (Soundness of the Inference Algorithm). *Let Γ and Δ be strongly consistent variable environments, M be a term, A be a type, σ be a substitution from type variables to types, and C, C' be sets of constraints. Then:*

(1) $(\Gamma, M : A \mid C \vdash \Delta) \in \mathit{InferL}(\Gamma, M, \Delta) \Rightarrow \forall \sigma.(\Gamma\sigma, M : A\sigma \mid C\sigma \vdash_2 \Delta\sigma)$
(2) $(\Gamma \mid C \vdash M : A) \in \mathit{InferR}(\Gamma, M) \Rightarrow \forall \sigma.(\Gamma\sigma \mid C\sigma \vdash_2 M : A\sigma)$

Proof. We proceed by induction on the shape of the term M.
Case(Specially treated rule):

In InferL there is a specially generated set that does not fall inline nicely with the others. The section of interest is:

$$\{\ \Gamma \cup \{\mathsf{M} : \mathsf{a}\} \mid \{\mathsf{Ok} \sqsubseteq \mathsf{b}\} \vdash \Delta \ \mid \\ \mathsf{a} = \mathsf{freshVar}, \\ (\mathsf{x} : \mathsf{b}) \in \Delta \\ \}$$

This set is the judgment(s) provable by (VarK2). This case is proven trivially but it is important to note that this judgment is produced by all terms M if the delta of the judgment is a variable. As this case is trivially sound, it will be ignored in future cases despite being generated in all of them.

Case(Var):

First, InferR. The section of interest is:

$$\{\ \Gamma \mid \{\mathsf{a} \sqsubseteq \mathsf{b}\} \vdash \mathsf{M} : \mathsf{b} \ \mid \\ \mathsf{b} = \mathsf{freshVar}, \\ \mathsf{a} = \begin{cases} t & (x : t) \in \Gamma \\ \mathsf{Ok} & \text{otherwise} \end{cases} \\ \} \cup \{\ \Gamma \mid \mathsf{C}[\overline{b}/\overline{a}] \cup \{\mathsf{a}[\overline{b}/\overline{a}] \sqsubseteq \mathsf{a'}\} \vdash \mathsf{M} : \mathsf{a'} \ \mid \\ (\mathsf{x} : \forall\ \overline{a}.\ \mathsf{C} \Rightarrow \mathsf{a}) \in \Gamma, \\ \overline{b} = \overline{\mathsf{freshVar}}, \\ \mathsf{a'} = \mathsf{freshVar} \\ \}$$

The second set creates a judgment for every typing of the variable at a (constrained) type scheme.

Fix a generated judgment, $\Gamma \mid C[\overline{b}/\overline{a}] \cup a[\overline{b}/\overline{a}] \sqsubseteq a' \vdash M : a'$.

Let σ be a type variable substitution and C' be the inferred constraint set, which is defined by $C[\overline{b}/\overline{a}] \cup a[\overline{b}/\overline{a}] \sqsubseteq a'$.

Then, by (Inst2), we have $\Gamma\sigma \mid C'\sigma \vdash_2 x : a'\sigma$.

The first set generates judgments for the (Var2) and (VarK2) rules.

Let σ be a type variable substitution.

By Γ being a strongly consistent variable environment, we have either $(x : t) \in \Gamma$ or $(x : t) \notin \Gamma$.

If $(x : t) \in \Gamma$ then, by (Var2), we have $\Gamma\sigma, x : t\sigma \mid \{t\sigma \sqsubseteq b\sigma\} \vdash_2 x : b\sigma$.

If $(x : t) \notin \Gamma$ then, by (VarK2), we have $\Gamma\sigma \mid \{\mathsf{Ok} \sqsubseteq b\sigma\} \vdash_2 x : b$.

Now, InferL. The section of interest is:

$$\{\ \Gamma \cup \{\mathsf{M} : \mathsf{a}\} \mid \{\mathsf{a} \sqsubseteq \mathsf{b}\} \vdash \Delta \ \mid \\ \mathsf{a} = \mathsf{freshVar}, \\ (\mathsf{x} : \mathsf{b}) \in \Delta \\ \}$$

Let σ be a type variable substitution. Then, by (Var2), we have that $\Gamma\sigma, x : a\sigma \mid \{a\sigma \sqsubseteq b\sigma\} \vdash_2 x : b\sigma$.

Case(App):

First, InferR. The section of interest is:

$$
\begin{aligned}
&\{\, \Gamma \mid c1 \cup c2 \cup \{b \sqsubseteq c \rightarrow\ a\} \vdash M : a \mid \\
&\quad (\Gamma \mid c1 \vdash P : b) \in \mathsf{inferR}(\Gamma, P), \\
&\quad (\Gamma \mid c2 \vdash Q : c) \in \mathsf{inferR}(\Gamma, Q), \\
&\quad a = \mathsf{freshVar} \\
&\}
\end{aligned}
$$

By the inductive hypothesis (applied to lines 2 and 3), we have:

(1) $\forall \sigma. \Gamma\sigma \mid C_1\sigma \vdash_2 P : b\sigma$
(2) $\forall \sigma. \Gamma\sigma \mid C_2\sigma \vdash_2 Q : c\sigma$

Let σ be a type variable substitution and the constraint set C be defined by $C_1\sigma \cup C_2\sigma \cup \{b\sigma \sqsubseteq c\sigma \rightarrow a\sigma\}$.
Then, by (AppR2), we have $\Gamma\sigma \mid C' \vdash_2 (P\ Q) : a\sigma$.

Now, InferL. The section of interest is:

$$
\begin{aligned}
&\{\, \Gamma \cup \{M : a\} \mid c1 \cup c2 \cup \{b \sqsubseteq c \rightarrowtail a\} \vdash \Delta \mid \\
&\quad a = \mathsf{freshVar}, \\
&\quad (\Gamma \mid c1 \vdash P : b) \in \mathsf{inferR}(\Gamma, P), \\
&\quad (\Gamma \cup \{Q : c\} \mid c2 \vdash d) \in \mathsf{inferL}(\Gamma, Q, \Delta) \\
&\} \cup \{\, \Gamma \cup \{M : a\} \mid c \cup \{\mathsf{Ok} \rightarrowtail a \sqsubseteq b\} \vdash \Delta \mid \\
&\quad a = \mathsf{freshVar}, \\
&\quad (\Gamma \cup \{P : b\} \mid c \vdash \Delta) \in \mathsf{inferL}(\Gamma, P, \Delta) \\
&\}
\end{aligned}
$$

The first set is the judgments provable by the (AppL2) rule.
By the inductive hypothesis (applied to lines 3 and 4), we have:

(1) $\forall \sigma. \Gamma\sigma \mid C_1\sigma \vdash_2 P : b\sigma$
(2) $\forall \sigma. \Gamma\sigma,\ Q : c\sigma \mid C_2\sigma \vdash_2 \Delta\sigma$

Let σ be a type variable substitution and the constraint set C be the inferred constraints defined by $C_1 \cup C_2 \cup \{b \sqsubseteq c \rightarrowtail a\}$.
Then, by (AppL2), we have $\Gamma\sigma,\ (P\ Q) : a\sigma \mid C\sigma \vdash_2 \Delta\sigma$.

The second set is the judgments provable by the (FunK2) rule.
By the inductive hypothesis (applied to line 7), we have: $\forall \sigma. \Gamma\sigma,\ P : b\sigma \mid C_1\sigma \vdash_2 \Delta\sigma$.
Let σ be a type variable substitution and constraint set C be the inferred constraints defined by $C_1 \cup \{\mathsf{Ok} \rightarrowtail a \sqsubseteq b\}$.
Then, by (FunK2), we have $\Gamma\sigma,\ (P\ Q) : a\sigma \mid C\sigma \vdash_2 \Delta\sigma$.
Case(Abs):

First, InferR. The section of interest is:

$$
\begin{array}{l}
\{\, \Gamma \mid c \cup \{t \rightarrow\ b \sqsubseteq a\} \vdash M : a \mid \\
\quad a = \mathsf{freshVar}, \\
\quad t = \mathsf{freshVar}, \\
\quad (\Gamma \cup \{x : t\} \mid c \vdash N : b) \in \mathsf{inferR}(\Gamma \cup \{x : t\}, N) \\
\} \cup \{\, \Gamma \mid c \cup \{t \succ\!\!- b \sqsubseteq a\} \vdash M : a \mid \\
\quad a = \mathsf{freshVar}, \\
\quad t = \mathsf{freshVar}, \\
\quad (\Gamma \cup \{N : b\} \mid c \vdash x : t) \in \mathsf{inferL}(\Gamma, N, \{x : t\}) \\
\}
\end{array}
$$

The first set is the judgments provable by the (AbsR2) rule.
By the inductive hypothesis (applied to line 4), we have $\forall\sigma.\Gamma\sigma,\ x : t\sigma \mid C_1\sigma \vdash_2 N : b\sigma$.
Let σ be a type variable substitution and constraint set C be the inferred constraints defined by $C_1 \cup \{t \rightarrow b \sqsubseteq a\}$.
Then, by (AbsR2), we have $\Gamma\sigma \mid C\sigma \vdash_2 \lambda x.\, n : a\sigma$.

The second set is the judgments provable by the (AbnR2) rule.
By the inductive hypothesis (applied to line 8), we have $\forall\sigma.\Gamma\sigma,\ N : b\sigma \mid C_1\sigma \vdash_2 x : t\sigma$.
Let σ be a type variable substitution and witness constraint set C be the inferred constraints defined by $C_1 \cup \{t \succ\!\!- b \sqsubseteq a\}$.
Then, by (AbnR2), we have $\Gamma\sigma \mid C\sigma \vdash_2 \lambda x.\, N : a\sigma$.

Now, InferL. The section of interest is:

$$
\begin{array}{l}
\{\, \Gamma \cup \{M : a\} \mid \{a \sqsubseteq \Sigma_{c \in C}(c(\overline{a}))\} \vdash \Delta \mid \\
\quad a = \mathsf{freshVar}, \\
\quad \overline{a} = \overline{\mathsf{freshVar}} \\
\}
\end{array}
$$

This case is trivially provable by (AbsDL2).

Case(Fix):
The only judgments produced for fix are in InferR. The section of interest is:

$$
\begin{array}{l}
\{\, \Gamma \mid c \cup \{b \sqsubseteq a\} \vdash M : a \mid \\
\quad a = \mathsf{freshVar}, \\
\quad (\Gamma \cup \{f : a\} \mid c \vdash m : b) \in \mathsf{inferR}(\Gamma \cup \{f : a\}, m) \\
\}
\end{array}
$$

By the inductive hypothesis (applied to line 3), we have $\forall\sigma.\Gamma\sigma,\ f : a\sigma \mid C_1\sigma \vdash_2 m : b\sigma$.
Let σ be a type variable substitution and constraint set C be the inferred constraints defined by $C_1 \cup \{b \sqsubseteq a\}$.
Then, by (FixR2), we have $\Gamma\sigma \mid C\sigma \vdash_2 \mathsf{fix}\, f.\, m : a\sigma$.

Case(Cons):

First, InferR. The section of interest is:

$$
\begin{aligned}
&\{\, \Gamma \mid \textstyle\bigcup_{i=1}^{n}(c_i) \cup \{\mathsf{c}(a_1,\ldots,a_n) \sqsubseteq \mathsf{a}\} \vdash \mathsf{M} : \mathsf{a} \;\mid \\
&\quad \mathsf{a} = \mathsf{freshVar}, \\
&\quad (\Gamma \mid c_i \vdash m_i : a_i)_{i=1}^{n} \in \Pi_{m \in \mathsf{ms}}(\mathsf{inferR}(\Gamma, \mathsf{m})) \\
&\} \backslash\mathsf{cup}\ \{\, \Gamma \mid \{\mathsf{c} \sqsubseteq \mathsf{a}\} \vdash \mathsf{M} : \mathsf{a} \;\mid \\
&\quad \mathsf{a} = \mathsf{freshVar}, \\
&\quad \mathsf{ms} = \{\,\} \\
&\}
\end{aligned}
$$

This is the set of judgments provable with the (CnsR2) rule.
The first set is for constructors with more than zero arguments (e.g. Cons and Succ), and the second set is for nullary constructors (e.g. Nil and Zero).
By the inductive hypothesis (applied to line 3), we have $\forall 1 \leq i \leq n. \forall \sigma. \Gamma\sigma \mid C_i\sigma \vdash_2 m_i : a_i\sigma$.
Let σ be a type variable substitution and constraint set C be the inferred constraints defined by $\bigcup_{i=1}^{n}(C_i) \cup \{c(a_1,\ldots,a_n) \sqsubseteq a\}$.
Then, for every combination of typings for the constructor's arguments, by (CnsR2), we have $\Gamma\sigma \mid C\sigma \vdash_2 c(M_1,\ldots,M_n) : a\sigma$.

The second set is proven sound trivially by (CnsR2).

Now, InferL. The section of interest is:

$$
\begin{aligned}
&\{\, \Gamma \cup \{\mathsf{M} : \mathsf{a}\} \mid \mathsf{c} \cup \{\mathsf{a} \sqsubseteq \Sigma_{\kappa' \in C \setminus \{\kappa\}}(\kappa'(\overline{a_{\kappa'}})) + \kappa(a_1, \ldots, b, \ldots, a_n)\} \vdash \Delta \;\mid \\
&\quad \mathsf{a} = \mathsf{freshVar},\ \overline{a_{\kappa'}} = \overline{\mathsf{freshVar}},\ (a)_{i=1}^{n} = \overline{\mathsf{freshVar}}, \\
&\quad m_i \in \mathsf{ms}, \\
&\quad (\Gamma \cup \{m_i : \mathsf{b}\} \mid \mathsf{c} \vdash \Delta) \in \mathsf{inferL}(\Gamma, \mathsf{m}, \Delta) \\
&\} \cup \{\, \Gamma \cup \{\mathsf{M} : \mathsf{a}\} \mid \mathsf{c} \cup \{\mathsf{Ok} \sqsubseteq \mathsf{b}\} \vdash \Delta \;\mid \\
&\quad \mathsf{a} = \mathsf{freshVar}, \\
&\quad m_i \in \mathsf{ms}, \\
&\quad (\Gamma \cup \{m_i : \mathsf{b}\} \mid \mathsf{c} \vdash \Delta) \in \mathsf{inferL}(\Gamma, \mathsf{m}, \Delta) \\
&\} \cup \{\, (\Gamma \cup \{\mathsf{M} : \mathsf{a}\} \mid \{a \sqsubseteq b_1 \to b_2\} \vdash \Delta) \mid \mathsf{a} = \mathsf{freshVar}, b_1 = \mathsf{freshVar}, b_2 = \mathsf{freshVar} \,\} \\
&\cup \{\, (\Gamma \cup \{\mathsf{M} : \mathsf{a}\} \mid \{a \sqsubseteq b_1 \rightarrowtail b_2\} \vdash \Delta) \mid \mathsf{a} = \mathsf{freshVar}, b_1 = \mathsf{freshVar}, b_2 = \mathsf{freshVar} \,\} \\
&\cup \{\, (\Gamma \cup \{\mathsf{M} : \mathsf{a}\} \mid \{\mathsf{a} \sqsubseteq \Sigma_{c \in C \setminus \kappa}(c(\overline{a}))\} \vdash \Delta) \mid \mathsf{a} = \mathsf{freshVar}, \overline{a} = \overline{\mathsf{freshVar}} \,\}
\end{aligned}
$$

These sets are the judgments provable by the: (CnsL2), (CnsK2), (CnsDL21), (CnsDL22), and (CnsDL23), respectively.
The last three sets are proven to be sound trivially by their respective rules.

For the first set, fix an inferred judgment $\Gamma,\ M : a \mid C_i \cup \{a \sqsubseteq \Sigma_{\kappa' \in C \setminus \{\kappa\}}(\kappa'(\overline{a_{\kappa'}})) + \kappa(a_1, \ldots, b, \ldots, a_n) \vdash \Delta\}$.
Then there exists an argument to the constructor, m_i such that $\Gamma,\ m_i : b \mid C_i \vdash \Delta$ is inferred by InferL(Γ, m_i, Δ).
By the inductive hypothesis, we have $\forall\sigma.\Gamma\sigma\ m_i : b\sigma \mid C_i\sigma \vdash_2 \Delta\sigma$.
Let σ be a type variable substitution and constraint set C be the inferred constraints defined by $C_i \cup \{a \sqsubseteq \Sigma_{\kappa' \in C \setminus \{\kappa\}}(\kappa'(\overline{a_{\kappa'}})) + \kappa(a_1, \ldots, b, \ldots, a_n)\}$.
Then, by (CnsL2), we have that $\Gamma\sigma,\ M : a\sigma \mid C\sigma \vdash_2 \Delta\sigma$.

For the second set, fix an inferred judgment $\Gamma,\ M : a \mid C_i \cup \{\mathsf{Ok} \sqsubseteq b\} \vdash \Delta$.
Then there exists an argument to the constructor, m_i such that $\Gamma,\ m_i : b \mid C_i \vdash \Delta$ is inferred by InferL(Γ, m_i, Δ).
By the inductive hypothesis, we have $\forall\sigma.\Gamma\sigma,\ m_i : b\sigma \mid C_i\sigma \vdash_2 \Delta\sigma$ and $C_i' \vdash C_i\sigma$.
Let σ be a type variable substitution and constraint set C be the inferred constraints defined by $C_i \cup \{\mathsf{Ok} \sqsubseteq b\}$.
Then, by (CnsK2), we have $\Gamma\sigma,\ M : a\sigma \mid C\sigma \vdash_2 \Delta\sigma$.

Case(Match):

First, InferR. The section of interest is:

$$\begin{aligned}
&\{\, \Gamma \mid \textstyle\bigcup_{i=1}^{k}(c_i \cup \{b \sqsubseteq p_i[b_x/x \mid x \in \mathsf{FV}(p_i)], a_i \sqsubseteq a\}) \cup \mathsf{c} \vdash \mathsf{M} : \mathsf{a} \;\mid \\
&\quad \mathsf{a} = \mathsf{freshVar}, \\
&\quad \overline{b_x} = \overline{\mathsf{freshVar}}, \\
&\quad (\Gamma \mid \mathsf{c} \vdash \mathsf{q} : \mathsf{b}) \in \mathsf{inferR}(\Gamma, \mathsf{Q}), \\
&\quad (\Gamma \cup \{\overline{(x : b_x)}\} \mid c_i \vdash P_i : a_i)_{i=1}^{k} \in \Pi_{i=1}^{k}(\mathsf{inferR}(\Gamma \cup \{\overline{(x : b_x)}\}, P_i)) \\
&\}
\end{aligned}$$

By the inductive hypothesis (applied to lines 4 and 5), we have:

(1) $\forall\sigma.\Gamma\sigma \mid C_0\sigma \vdash_2 q : b\sigma$
(2) $\forall 1 \le i \le n.\forall\sigma.\Gamma\sigma, \overline{(x : b')} \mid C_i\sigma \vdash_2 p_i : a_i\sigma$

Let σ be a type variable substitution and constraint set C be the inferred constraints defined by $\bigcup_{i=1}^{k}(C_i \cup \{b \sqsubseteq p_i[b_x/x \mid x \in \mathsf{FV}(p_i)], a_i \sqsubseteq a\}) \cup C_0$.
Then, for all combinations of typings of case bodies, by (MchR2), we have $\Gamma\sigma \mid C\sigma \vdash_2 M : a\sigma$.
Now, InferL. The section of interest is:

$$\begin{aligned}
&\{\, \Gamma \cup \{\mathsf{M} : \mathsf{a}\} \mid \mathsf{c} \vdash \Delta \;\mid \\
&\quad \mathsf{a} = \mathsf{freshVar}, \\
&\quad (a_i, b_i)_{i=1}^{k} = \overline{\mathsf{freshVar}}, \\
&\quad (b_{(i,x)})_{(i,x)\in[1..k]\times\mathsf{FV}(p_i)} = \overline{\mathsf{freshVar}}, \\
&\quad (\Gamma \cup \{(\mathsf{q}, P_i) : a_i'\} \mid c_i \vdash \Delta)_{i=1}^{k} \in \Pi_{i=1}^{k}(\mathsf{inferL}(\Gamma, (\mathsf{Q}, P_i), \Delta)), \\
&\quad (\Gamma \cup \{P_i : a_{(i,x)}\} \mid c'_{(i,x)} \vdash \Delta)_{(i,x)} \in \Pi_{(i,x)\in[1..k]\times\mathsf{FV}(p_i)}(\mathsf{inferL}(\Gamma, P_i, \{\mathsf{x} : b_{(i,x)}\})), \\
&\quad \mathsf{c} = \textstyle\bigcup_{i=1}^{k}(c_i \cup \{a \sqsubseteq a_i, (a_i, b_i) \sqsubseteq a_i', p_i[b_{(i,x)}/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq b_i\}) \cup \\
&\qquad \textstyle\bigcup_{(i,x)\in[1..k]\times\mathsf{FV}(p_i)}(c'_{(i,x)} \cup \{a \sqsubseteq a_{(i,x)}\}) \\
&\}
\end{aligned}$$

By the inductive hypothesis (applied to lines 5 and 6), we have:

(1) $\forall 1 \le i \le k.\forall\sigma.\Gamma\sigma, (Q, P_i) : a_i'\sigma \mid C_i\sigma \vdash_2 \Delta\sigma$
(2) $\forall 1 \le i \le k.\forall x \in \mathsf{FV}(p_i).\forall\sigma.\Gamma\sigma, P_i : a_{(i,x)}\sigma \mid C'_{(i,x)}\sigma \vdash_2 \Delta\sigma$

Let σ be a type variable substitution and constraint set C be defined by:

$$\begin{aligned}
C := &\bigcup_{i=1}^{k}(C_i \cup \{a \sqsubseteq a_i, (a_i, b_i) \sqsubseteq a_i', p_i[b_{(i,x)}/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq b_i\}) \cup \\
&\bigcup_{(i,x)\in[1..k]\times\mathsf{FV}(p_i)}(C'_{(i,x)} \cup \{a \sqsubseteq a_{(i,x)}\})
\end{aligned}$$

Then, for all combinations of typings, by (MchL2), we have $\Gamma\sigma, M : a\sigma \mid C\sigma \vdash_2 \Delta\sigma$.

□

Theorem B.5 (Completeness of the Inference Algorithm). *Let τ, σ be substitutions from type variables to types such that σ extends τ, $\Gamma\tau, \Delta\tau$ be strongly consistent variable environments, M be a term, A be a type, and C, C' be sets of constraints. Then:*

(1) $\Gamma\tau, M : A \mid C \vdash_2 \Delta\tau \Rightarrow \exists\sigma.\exists(\Gamma, M : A' \mid C' \vdash \Delta) \in \mathit{InferL}(\Gamma, M, \Delta).\ (A = A'\sigma) \wedge (C \vdash C'\sigma)$
(2) $\Gamma\tau \mid C \vdash_2 M : A \Rightarrow \exists\sigma.\exists(\Gamma \mid C' \vdash M : A') \in \mathit{InferR}(\Gamma, M).\ (A = A'\sigma) \wedge (C \vdash C'\sigma)$

Proof. We proceed by induction on the derivation of $\vdash_2$.
Case(Inst2):

For some τ, we have $\Gamma\tau$, $f : \forall\vec{a}.C' \Rightarrow A \mid C \vdash_2 f : A', C \vdash C'[\vec{B}/\vec{a}]$, and $C \vdash A[\vec{B}/\vec{a}] \sqsubseteq A'$.
Observe:

$$
\begin{array}{l}
\{\ \mathsf{g} \mid \mathsf{c}[\overline{B}/\overline{a}] \cup \{\mathsf{A}[\overline{B}/\overline{a}] \sqsubseteq \mathsf{a'}\} \vdash \mathsf{M} : \mathsf{a'} \ \mid \\
\quad (\mathsf{x} : \forall\ \overline{a}.\ \mathsf{c} \Rightarrow \mathsf{A}) \in \mathsf{g}, \\
\quad \overline{B} = \overline{\mathsf{freshVar}}, \\
\quad \mathsf{a'} = \mathsf{freshVar} \\
\}
\end{array}
$$

Let σ be a type variable substitution defined by $\tau \cup [\overline{B} \mapsto \vec{B}, a' \mapsto A'$. As the inferred set is a singleton, the witness judgment is clear. Then, by definition of σ, we have $A' = a'\sigma$ and $C \vdash (C'[\overline{B}/\overline{a}] \cup \{A[\overline{B}/\overline{a}] \sqsubseteq a'\})\sigma$, as required.

Case(Var2):
For some τ, we have $\Gamma\tau$, $x : A \mid C \vdash_2 x : B$.
By the symmetry of the rule, we need to show both cases of the the rule (both InferL and InferR).
In InferR, observe:

$$
\begin{array}{l}
\{\ \mathsf{g} \mid \{\mathsf{a} \sqsubseteq \mathsf{b}\} \vdash \mathsf{M} : \mathsf{b} \ \mid \\
\quad \mathsf{b} = \mathsf{freshVar}, \\
\quad \mathsf{a} = \begin{cases} t & (x : t) \in g \\ \mathsf{Ok} & \text{otherwise} \end{cases} \\
\}
\end{array}
$$

As $x : A$ is in the context, the variable on line 3 is $a = A$. So the inferred judgment is: $\Gamma, x : A \mid \{A \sqsubseteq b\} \vdash x : b$. Thus, taking $\sigma := \tau \cup [b \mapsto B]$ trivially satisfies the requirements.
In InferL, observe:

$$
\begin{array}{l}
\{\ \mathsf{g} \cup \{\mathsf{M} : \mathsf{a}\} \mid \{\mathsf{a} \sqsubseteq \mathsf{b}\} \vdash \mathsf{d} \ \mid \\
\quad \mathsf{a} = \mathsf{freshVar}, \\
\quad (\mathsf{x} : \mathsf{b}) \in \mathsf{d} \\
\}
\end{array}
$$

As $x : B$ is in Δ, the judgment inferred is: $\Gamma, x : a \mid \{a \sqsubseteq B\} \vdash x : B$. Thus, taking $\sigma := \tau \cup [a \mapsto A]$ trivially satisfies the requirements.

Case(VarK2):

For some τ, we have $\Gamma\tau \mid C \vdash_2 x : A$ and $C \vdash \mathsf{Ok} \sqsubseteq A$.
As this case only requires a variable typing in the delta position, the non-trivial term can be either on the left or the right of the judgment.
In InferR, observe:

$$\{\, \mathsf{g} \mid \{\mathsf{a} \sqsubseteq \mathsf{b}\} \vdash \mathsf{M} : \mathsf{b} \;\mid\; \mathsf{b} = \mathsf{freshVar},\; \mathsf{a} = \begin{cases} t & (x : t) \in g \\ \mathsf{Ok} & \text{otherwise} \end{cases} \,\}$$

If $(x : t) \in \Gamma$ then the inferred judgment is: $\Gamma \mid \{t \sqsubseteq b\} \vdash x : b$. Taking $\sigma := \tau \cup [b \mapsto A]$ trivially satisfies the requirements.
If $(x : t) \notin \Gamma$ then the inferred judgment is: $\Gamma \mid \{\mathsf{Ok} \sqsubseteq b\} \vdash x : b$. Taking $\sigma := \tau \cup [b \mapsto A]$ trivially satisfies the requirements.

In InferL, observe:

$$\{\, \mathsf{g} \cup \{\mathsf{M} : \mathsf{a}\} \mid \{\mathsf{Ok} \sqsubseteq \mathsf{b}\} \vdash \mathsf{d} \;\mid\; \mathsf{a} = \mathsf{freshVar},\; (\mathsf{x} : \mathsf{b}) \in \mathsf{d} \,\}$$

As $x : A$ is the delta, the inferred judgment (for all terms M) is: $\Gamma,\ M : a \mid \{\mathsf{Ok} \sqsubseteq b\} \vdash x : b$.
As $M : a$ is included in the context of the judgment, $a\tau$ is a concrete type.
Thus, taking $\sigma := \tau \cup [b \mapsto A]$ trivially satisfies the requirements.

Case(AbsR2):
For some τ, we have $\Gamma\tau \mid C \vdash_2 \lambda x.\, M : A,\ \ \Gamma\tau,\ x : B_1 \mid C \vdash_2 M : B_2$ and $C \vdash B_1 \to B_2 \sqsubseteq A$.
By the inductive hypothesis, there exists a σ that extends τ and an inferred judgment $(\Gamma,\ x : b_1 \mid C_0' \vdash_2 M : b_2) \in \mathrm{InferR}(\Gamma \cup \{x : b_1\},\ M)$ such that $B_2 = b_2\sigma$ and $C \vdash C_0'\sigma$.
Observe this extract from InferR in the abstraction case:

$$\{\, \mathsf{g} \mid \mathsf{c} \cup \{\mathsf{t} \to \mathsf{b} \sqsubseteq \mathsf{a}\} \vdash \mathsf{M} : \mathsf{a} \;\mid\; \mathsf{a} = \mathsf{freshVar},\; \mathsf{t} = \mathsf{freshVar},\; (\mathsf{g} \cup \{\mathsf{x} : \mathsf{t}\} \mid \mathsf{c} \vdash \mathsf{n} : \mathsf{b}) \in \mathsf{inferR}(\mathsf{g} \cup \{\mathsf{x} : \mathsf{t}\},\ \mathsf{n}) \,\}$$

With this inferred judgment, extending σ to the witness $\sigma' = \sigma \cup [b_1 \mapsto B_1, a \mapsto A]$ clearly satisfies the requirements of $A = a\sigma'$ and $C \vdash (C_0' \cup \{b_1 \to b_2 \sqsubseteq a\})\sigma'$.

Case(AbnR2):

For some τ, we have $\Gamma\tau \mid C \vdash_2 \lambda x.\, M : A,\ \Gamma\tau,\, M : B_1 \mid C \vdash_2 x : B_2$ and $C \vdash B_2 \rightarrowtail B_1 \sqsubseteq A$.
By the inductive hypothesis, there exists a σ that extends τ and an inferred judgment $(\Gamma,\, M : b_1 \mid C_0' \vdash_2 x : b_2) \in \mathrm{InferL}(\Gamma,\, M,\, \{x : b_2\})$ such that $B_1 = b_1\sigma$ and $C \vdash C_0'\sigma$.
Observe this extract from InferR in the abstraction case:

```
{ g | c ∪ {t ↣ b ⊑ a} ⊢ M : a  |
    a = freshVar,
    t = freshVar,
    (g ∪ {n : b} | c ⊢ x : t) ∈ inferL(g, n, {x : t})
}
```

With this inferred judgment, extending σ to the witness $\sigma' = \sigma \cup [b_2 \mapsto B_2, a \mapsto A]$ clearly satisfies the requirements of $A = a\sigma'$ and $C \vdash (C_0' \cup \{b_2 \rightarrowtail b_1 \sqsubseteq a\})\sigma'$.

Case(AbsDL2):

For some τ, we have $\Gamma\tau,\, \lambda x.\, M : A \mid C \vdash_2 \Delta\tau$ and $C \vdash A \sqsubseteq \Sigma_{c \in C}(c(\vec{A_c}))$.
Observe this extract from InferL in the abstraction case:

```
{ g ∪ {M : a} | {a ⊑ Σ_{c∈C}(c(ā))} ⊢ d  |
    a = freshVar,
    ā = freshVar
}
```

Define the witness σ by $\sigma := \tau \cup [\overline{a} \mapsto \vec{A_c}, a \mapsto A]$.
Then, $A = a\sigma$ and $\{a \sqsubseteq \Sigma_{c \in C}(c(\overline{a}))\}\sigma = \{A \sqsubseteq \Sigma_{c \in C}(c(\vec{A_c}))\}$ and so $C \vdash C'\sigma$ as required.

Case(FixR2):

For some τ, we have $\Gamma\tau \mid C \vdash_2 \mathrm{fix}\, f.\, M : A,\ \Gamma\tau,\, f : A \mid C \vdash_2 M : B$ and $C \vdash B \sqsubseteq A$.
By the inductive hypothesis, there exists a σ that extends τ and an inferred judgment $(\Gamma,\, f : a \mid C_0' \vdash_2 M : b) \in \mathrm{InferR}(\Gamma \cup \{f : a\},\, M)$ such that $B = b\sigma$ and $C \vdash C_0'\sigma$.
Observe this extract from InferR in the fix-point case:

```
{ g | c ∪ {b ⊑ a} ⊢ M : a  |
    a = freshVar,
    (g ∪ {f : a} | c ⊢ m : b) ∈ inferR(g ∪ {f : a}, m)
}
```

With this inferred judgment, extending σ to the witness $\sigma' = \sigma \cup [a \mapsto A]$ clearly satisfies the requirements of $A = a\sigma'$ and $C \vdash (C_0 \cup \{b \sqsubseteq a\})\sigma'$.

Case(AppR2):

For some τ, we have $\Gamma\tau \mid C \vdash_2 (l\ r) : A$, $\Gamma\tau \mid C \vdash_2 l : B_1$, $\Gamma\tau \mid C \vdash_2 r : B_2$, and $C \vdash B_1 \sqsubseteq B_2 \to A$.

By the inductive hypothesis, we have:

(1) $\exists\sigma_1.\exists(\Gamma \mid C_1' \vdash l : b_1) \in \text{InferR}(\Gamma, l).B_1 = b_1\sigma_1 \wedge C \vdash C_1'\sigma_1$
(2) $\exists\sigma_2.\exists(\Gamma \mid C_2' \vdash r : b_2) \in \text{InferR}(\Gamma, r).B_2 = b_2\sigma_2 \wedge C \vdash C_2'\sigma_2$

Observe this extract from InferR in the application case:

```
{ g | c1 ∪ c2 ∪ {b ⊑ c → a} ⊢ M : a |
    (g | c1 ⊢ l : b) ∈ inferR(g, l),
    (g | c2 ⊢ r : c) ∈ inferR(g, r),
    a = freshVar
}
```

With this inferred judgment, and the observation that σ_1 and σ_2 only share variables given by τ (i.e. $\sigma_1 \cap \sigma_2 = \tau$), we define a witness $\sigma = \sigma_1 \cup \sigma_2 \cup [a \mapsto A]$. Thus, $A = a\sigma$ and $C \vdash (C_1 \cup C_2 \cup \{b_1 \sqsubseteq b_2 \to a\})\sigma$ as required.

Case(AppL2):

For some τ, we have $\Gamma\tau$, $(l\ r) : A \mid C \vdash_2 \Delta\tau$, $\Gamma\tau \mid C \vdash_2 l : B_1$, $\Gamma\tau, r : B_2 \mid C \vdash_2 \Delta\tau$, and $C \vdash B_1 \sqsubseteq B_2 \rightarrowtail A$.

By the inductive hypothesis, we have:

(1) $\exists\sigma_1.\exists(\Gamma \mid C_1' \vdash l : b_1) \in \text{InferR}(\Gamma, l).B_1 = b_1\sigma_1 \wedge C \vdash C_1'\sigma_1$
(2) $\exists\sigma_2.\exists(\Gamma, r : b_2 \mid C_2' \vdash \Delta) \in \text{InferL}(\Gamma, r, \Delta).B_2 = b_2\sigma_2 \wedge C \vdash C_2'\sigma_2$

Observe this extract from InferL in the application case:

```
{ g ∪ {M : a} | c1 ∪ c2 ∪ {b ⊑ c ↣ a} ⊢ d |
    a = freshVar,
    (g | c1 ⊢ l : b) ∈ inferR(g, l),
    (g ∪ {r : c} | c2 ⊢ d) ∈ inferL(g, r, d)
}
```

With this inferred judgment, and the observation that σ_1 and σ_2 only share variables given by τ (i.e. $\sigma_1 \cap \sigma_2 = \tau$), we define a witness $\sigma = \sigma_1 \cup \sigma_2 \cup [a \mapsto A]$. Thus, $A = a\sigma$ and $C \vdash (C_1 \cup C_2 \cup \{b_1 \sqsubseteq b_2 \rightarrowtail a\})\sigma$ as required.

Case(AppK2):

For some τ, we have $\Gamma\tau$, $(l\ r) : A \mid C \vdash_2 \Delta\tau$, $\Gamma\tau, l : B \mid C \vdash_2 \Delta\tau$, and $C \vdash \text{Ok} \rightarrowtail A \sqsubseteq B$.

By the inductive hypothesis, there exists a σ_1 extending τ and an inferred judgment $(\Gamma, l : b \mid C_1' \vdash \Delta) \in \text{InferL}(\Gamma, l, \Delta)$ such that $B = b\sigma_1$ and $C \vdash C_1'\sigma_1$.

Observe this extract from InferL in the application case:

```
{ g ∪ {M : a} | c ∪ {Ok ↣ a ⊑ b} ⊢ d |
    a = freshVar,
    (g ∪ {l : b} | c ⊢ d) ∈ inferL(g, l, d)
}
```

With this inferred judgment, we define a witness $\sigma = \sigma_1 \cup [a \mapsto A]$. Thus, $A = a\sigma$ and $C \vdash (C_1 \cup \{\text{Ok} \rightarrowtail a \sqsubseteq b\})\sigma$ as required.

Case(CnsDL):

All of the (CnsDL21), (CnsDL22), and (CnsDL23) cases are all very similar.
Observe this extract from InferL in the constructor case:

$$
\begin{array}{l}
\{\, \mathsf{g} \cup \{\mathsf{M} : \mathsf{a}\} \mid \{a \sqsubseteq b_1 \to b_2\} \vdash \mathsf{d} \;\mid \\
\quad \mathsf{a} = \mathsf{freshVar}, \\
\quad b_1 = \mathsf{freshVar}, \\
\quad b_2 = \mathsf{freshVar} \\
\} \cup \{\, \mathsf{g} \cup \{\mathsf{M} : \mathsf{a}\} \mid \{a \sqsubseteq b_1 \rightarrowtail b_2\} \vdash \mathsf{d} \;\mid \\
\quad \mathsf{a} = \mathsf{freshVar}, \\
\quad b_1 = \mathsf{freshVar}, \\
\quad b_2 = \mathsf{freshVar} \\
\} \cup \{\, \mathsf{g} \cup \{\mathsf{M} : \mathsf{a}\} \mid \{\mathsf{a} \sqsubseteq \Sigma_{c \in C \setminus \kappa}(c(\overline{a}))\} \vdash \mathsf{d} \;\mid \\
\quad \mathsf{a} = \mathsf{freshVar}, \\
\quad \overline{a} = \overline{\mathsf{freshVar}} \\
\}
\end{array}
$$

Each of the inferred sets above are singletons and correspond to the (CnsDL21), (CnsDL22), and (CnsDL23) rules, respectively. Taking the respective witnesses to be defined by:

(1) $\sigma := \tau \cup [b_1 \mapsto B_1, b_2 \mapsto B_2, a \mapsto A]$
(2) $\sigma := \tau \cup [b_1 \mapsto B_1, b_2 \mapsto B_2, a \mapsto A]$
(3) $\sigma := \tau \cup [\overline{a} \mapsto \vec{a}, a \mapsto A]$

Each of the witness substitutions and judgments trivially satisfy the requirements.

Case(CnsR2):
For some τ, we have $\Gamma\tau \mid C \vdash_2 c(M_1, \ldots, M_n) : A$, $\forall 1 \leq i \leq n. \Gamma\tau \mid C \vdash_2 M_i : A_i$, and $C \vdash c(A_1, \ldots, A_n) \sqsubseteq A$.
By the inductive hypothesis, for all $1 \leq i \leq n$, there exists a σ_i extending τ and an inferred judgment $(\Gamma \mid C_i' \vdash M_i : a_i) \in \mathrm{InferR}(\Gamma, M_i)$ such that $A_i = a_i\sigma_i$ and $C \vdash C_i'\sigma_i$.
Observe this extract from InferR in the constructor case:

$$
\begin{array}{l}
\{\, \mathsf{g} \mid \bigcup_{i=1}^{n}(c_i) \cup \{\mathsf{c}(a_1, \ldots, a_n) \sqsubseteq \mathsf{a}\} \vdash \mathsf{M} : \mathsf{a} \;\mid \\
\quad \mathsf{a} = \mathsf{freshVar}, \\
\quad (\mathsf{g} \mid c_i \vdash m_i : a_i)_{i=1}^{n} \in \Pi_{m \in \mathsf{ms}}(\mathsf{inferR}(\mathsf{g}, \mathsf{m})) \\
\}
\end{array}
$$

As there is clearly a tuple of length n with the collection of inferred judgments that correspond to those from the inductive hypothesis on line 3, we take the inferred judgment $\Gamma \mid \bigcup_{i=1}^{n}(C_i') \cup \{c(a_1, \ldots, a_n) \sqsubseteq a\} \vdash c(M_1, \ldots, M_n) : a$ as the required witness. We then define the witness $\sigma := \bigcup_{i=1}^{n}(\sigma_i) \cup [a \mapsto A]$, which is well defined as $\sigma_i \cap \sigma_j = \tau$ for all $i \neq j$ by each recursive call creating fresh variables. Thus, $A = a\sigma$ and $C \vdash (\bigcup_{i=1}^{n}(C_i') \cup \{c(a_1, \ldots, a_n) \sqsubseteq a\})\sigma$ as required.

Case(CnsL2):

For some τ, we have $\Gamma\tau$, $c(M_1, \ldots, M_n) : A \mid C \vdash_2 \Delta\tau$, $\exists i.\Gamma\tau, M_i : A_i \mid C \vdash_2 \Delta\tau$, and $C \vdash A \sqsubseteq \Sigma_{d \in C \setminus \{c\}} + c(A_1, \ldots, A_n)$.
By the inductive hypothesis, there exists some $1 \leq i \leq n$ such that, there exists a σ_i extending τ and an inferred judgment $(\Gamma, M_i : a_i \mid C_i' \vdash \Delta) \in \text{InferL}(\Gamma, M_i, \Delta)$ such that $A_i = a_i\sigma_i$ and $C \vdash C_i'\sigma_i$.
Observe this extract from InferL in the constructor case:

```
{ g ∪ {M : a} | c ∪ {a ⊑ Σ_{κ' ∈ C\{κ}}(κ'(a_κ')) + κ(a_1, ..., b, ..., a_n)} ⊢ d |
    a = freshVar,
    a_κ' = freshVar,
    (a)_{i=1}^n = freshVar,
    m_i ∈ ms,
    (g ∪ {m_i : b} | c ⊢ d) ∈ inferL(g, m, d)
}
```

Line 5 shows that an inferred judgment is generated for all i, thus it is possible to pick the correct i and correct judgment in line 6 to produce the required witness judgment.
Defining the witness $\sigma := \sigma_i \cup [\overline{a_\kappa'} \mapsto \vec{A_c}, (a_i)_{i \in [1..n] \setminus \{i\}} \mapsto (A_i)_{i \in [1..n] \setminus \{i\}}, b \mapsto A_i, a \mapsto A]$ satisfies the requirements of $A = a\sigma$ and $C \vdash (C_i' \cup \{a \sqsubseteq \Sigma_{\kappa' \in C \setminus \{\kappa\}}(\kappa'(\overline{a_{\kappa'}})) + \kappa(a_1, \ldots, b, \ldots, a_n)\})\sigma$ as required.

Case(CnsK2):

For some τ, we have $\Gamma\tau$, $c(M_1, \ldots, M_n) : A \mid C \vdash_2 \Delta\tau$, $\exists i.\Gamma\tau, M_i : B \mid C \vdash_2 \Delta\tau$, and $C \vdash \text{Ok} \sqsubseteq B$.
By the inductive hypothesis, there exists some $1 \leq i \leq n$ such that, there exists a σ_i extending τ and an inferred judgment $(\Gamma, M_i : b \mid C_i' \vdash \Delta) \in \text{InferL}(\Gamma, M_i, \Delta)$ such that $B = b\sigma_i$ and $C \vdash C_i'\sigma_i$.
Observe this extract from InferL in the constructor case:

```
{ g ∪ {M : a} | c ∪ {Ok ⊑ b} ⊢ d |
    a = freshVar,
    m_i ∈ ms,
    (g ∪ {m_i : b} | c ⊢ d) ∈ inferL(g, m, d)
}
```

Line 3 shows that an inferred judgment is generated for all i, thus it is possible to pick the correct i and correct judgment in line 4 to produce the required witness judgment.
Defining the witness $\sigma := \sigma_i \cup [a \mapsto A]$ satisfies the requirements of $A = a\sigma$ and $C \vdash (C_i' \cup \{Ok \sqsubseteq b\})\sigma$.

Case(MchR2):

For some τ, we have:

(1) $\Gamma\tau \mid C \vdash_2 \mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}(p_i \mapsto P_i)\} : A$
(2) $\gamma\tau \mid C \vdash_2 Q : B$
(3) $\Gamma\tau \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \mid C \vdash_2 P_i : A_i\ (\forall i)$
(4) $C \vdash B \sqsubseteq \Sigma_{i=1}^{k}(p_i[B_x/x \mid x \in \mathsf{FV}(p_i)])$
(5) $\forall i. C \vdash A_i \sqsubseteq A$

By the inductive hypothesis, we have:

(1) $\exists\sigma_0.\exists(\Gamma \mid C_0' \vdash Q : b) \in \mathrm{InferR}(\Gamma, Q).B = b\sigma_0 \wedge C \vdash C_0'\sigma_0$
(2) $\forall 1 \le i \le k.\exists\sigma_i.\exists(\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \mid C_i' \vdash P_i : a_i) \in \mathrm{InferR}(\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\}, P_i).A_i = a_i\sigma_i \wedge C \vdash C_i'\sigma_2$

Observe this extract from InferR in the match case:

$$\begin{array}{l}\{\, \mathsf{g} \mid \bigcup_{i=1}^{k}(c_i \cup \{b \sqsubseteq p_i[b_x/x \mid x \in \mathsf{FV}(p_i)], a_i \sqsubseteq a\}) \cup \mathsf{c} \vdash \mathsf{M} : \mathsf{a} \mid \\ \quad \mathsf{a} = \mathsf{freshVar}, \\ \quad \overline{b_x} = \overline{\mathsf{freshVar}}, \\ \quad (\mathsf{g} \mid \mathsf{c} \vdash \mathsf{q} : \mathsf{b}) \in \mathsf{inferR}(\mathsf{g}, \mathsf{q}), \\ \quad (\mathsf{g} \cup \{\overline{(x : b_x)}\} \mid c_i \vdash P_i : a_i)_{i=1}^{k} \in \Pi_{i=1}^{k}(\mathsf{inferR}(\mathsf{g} \cup \{\overline{(x : b_x)}\}, P_i)) \\ \}\end{array}$$

By choosing the correct inferred judgment(s) in lines 4 and 5 makes it clear that the correct witness judgment can be inferred.

Defining the witness $\sigma := \sigma_0 \cup \bigcup_{i=1}^{k}(\sigma_i \cup [(b_x)_{x \in \mathsf{FV}(p_i)} \mapsto (B_x)_{x \in \mathsf{FV}(p_i)}]) \cup [a \mapsto A]$ satisfies the requirements of $A = a\sigma$ and $C \vdash (\bigcup_{i=1}^{k}(C_i' \cup \{b \sqsubseteq p_i[b_x/x \mid x \in \mathsf{FV}(p_i)], a_i \sqsubseteq a\}) \cup C_0')\sigma$.

Case(MchL2):

For some τ, we have:

(1) $\Gamma\tau,\ \text{match } Q \text{ with } \{|_{i=1}^{k}(p_i \mapsto P_i)\} : A \mid C \vdash_2 \Delta\tau$
(2) $\forall i.\Gamma\tau,\ (M, P_i) : A'_i \mid C \vdash_2 \Delta\tau$
(3) $\forall i.\forall x \in \mathsf{FV}(p_i).\Gamma,\ P_i : A_x \mid C \vdash_2 x : B_x$
(4) $\forall i.C \vdash A \sqsubseteq A_i$
(5) $\forall i.C \vdash (B_i, A_i) \sqsubseteq A'_i$
(6) $\forall i.\forall x \in \mathsf{FV}(p_i).C \vdash A \sqsubseteq A_x$
(7) $\forall i.C \vdash p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq B_i$

By the inductive hypothesis, we have:

(1) $\forall i.\exists\sigma'_i.\exists(\Gamma,\ (M, P_i) : a'_i \mid C'_i \vdash \Delta) \in \mathrm{InferL}(\Gamma,\ (M, P_i),\ \Delta).A'_i = a'_i\sigma'_i \wedge C \vdash C'_i\sigma'_i$
(2) $\forall i.\forall x \in \mathsf{FV}(p_i).\exists\sigma_{(i,x)}.\exists(\Gamma,\ P_i : a_x \mid C_{(i,x)} \vdash x : b_x) \in \mathrm{InferL}(\Gamma,\ P_i,\ \{x : b_x\}).A_x = a_x\sigma_{(i,x)} \wedge C \vdash C_{(i,x)}\sigma_{(i,x)}$

Observe this extract from InferL in the match case:

$$
\begin{array}{l}
\{\ \mathsf{g} \cup \{\mathsf{M} : \mathsf{a}\} \mid \mathsf{c} \vdash \mathsf{d}\ \mid \\
\quad \mathsf{a} = \mathsf{freshVar}, \\
\quad (a_i, b_i)_{i=1}^{k} = \overline{\mathsf{freshVar}}, \\
\quad (b_{(i,x)})_{(i,x)\in[1..k]\times\mathsf{FV}(p_i)} = \overline{\mathsf{freshVar}}, \\
\quad (\mathsf{g} \cup \{(\mathsf{q}, P_i) : a'_i\} \mid c_i \vdash \mathsf{d})_{i=1}^{k} \in \Pi_{i=1}^{k}(\mathsf{inferL}(\mathsf{g}, (\mathsf{q}, P_i), \mathsf{d})), \\
\quad (\mathsf{g} \cup \{P_i : a_{(i,x)}\} \mid c'_{(i,x)} \vdash \mathsf{d})_{(i,x)} \in \Pi_{(i,x)\in[1..k]\times\mathsf{FV}(p_i)}(\mathsf{inferL}(\mathsf{g}, P_i, \{\mathsf{x} : b_{(i,x)}\})), \\
\quad \mathsf{c} = \bigcup_{i=1}^{k}(c_i \cup \{a \sqsubseteq a_i, (a_i, b_i) \sqsubseteq a'_i, p_i[b_{(i,x)}/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq b_i\}) \cup \\
\qquad \bigcup_{(i,x)\in[1..k]\times\mathsf{FV}(p_i)}(c'_{(i,x)} \cup \{a \sqsubseteq a_{(i,x)}\}) \\
\}
\end{array}
$$

Line 5 generates a tuple of length k by enumerating every arrangement of typings for the k pairs (Q, P_i). Hence, the correct sequence of types can be picked according to the inductive hypothesis (point 1).
Line 6 generates a tuple by enumerating every arrangement of typings for each (left) typing of P_i with delta $x : b_{(i,x)}$ (for each i and x). Hence, the correct sequence of types can be picked according to the inductive hypothesis (point 2).
Hence, the required witness judgment is generated.
As each recursive call produces its own, independent, fresh variables, the pairwise intersection of any two distinct σ given by the inductive hypotheses is τ, we can define the witness:

$$\sigma := \bigcup_{i=1}^{k}(\sigma_i \cup [a_i \mapsto A_i, b_i \mapsto B_i] \cup \bigcup_{x\in\mathsf{FV}(p_i)}(\sigma_{(i,x)} \cup [b_{(i,x)} \mapsto B_x])) \cup [a \mapsto A]$$

This satisfies $A = a\sigma$ and $C \vdash C'\sigma$ (where C' is the generated constraints on line 7/8 of the extract) as required.

□

C ADDITIONAL MATERIAL IN SUPPORT OF SUBSECTION 5.3

In this appendix, we give the definition of reduction for the programming language of the constrained system and we give the definition of consistency of a constraint set. We then establish a number of useful lemmas regarding the type system:

- left and right inversion lemmas
- typed substitution lemmas

This enables us to prove progress and preservation, and finally conclude syntactic soundness.

Definition C.1 (Reduction). The *evaluation contexts* are defined by the following grammar:

$$\mathcal{E} ::= \Box \mid c(\vec{V}, \mathcal{E}, \vec{M}) \mid \mathcal{E}\, N \mid (\lambda x.\, M)\, \mathcal{E} \mid \mathsf{match}\, \mathcal{E}\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$$

Fix a module $\mathcal{M}$, then the one-step reduction relation wrt $\mathcal{M}$, written $M \rhd_{\mathcal{M}} N$, is the contextual closure of the following axiom schema:

$$\text{(Delta)}\quad \mathsf{f} \rhd_{\mathcal{M}} \mathcal{M}(\mathsf{f}) \qquad \text{(Beta)}\quad (\lambda x.\, M)\, V \rhd_{\mathcal{M}} M[V/x] \qquad \text{(Fix)}\quad \mathsf{fix}\, x.\, M \rhd_{\mathcal{M}} M[\mathsf{fix}\, x.\, M/x]$$

$$\text{(Match)}\quad \mathsf{match}\, c(\vec{V})\, \mathsf{with}\, \{|_{d \in I}(d(\vec{x_d}) \mapsto P_d)\} \rhd_{\mathcal{M}} P_c[\overrightarrow{V/x_c}] \qquad (\text{when } c \in I)$$

We will more often simply write $M \rhd N$ whenever the context is unimportant or otherwise understood, and the reflexive-transitive closure by $M \rhd^* N$.

Definition C.2. A constraint set C is said to be *closed* just if the following conditions are met:

(1) If $A \sqsubseteq A' \in C$ and $A' \sqsubseteq B \in C$ then $A \sqsubseteq B \in C$.
(2) If $K \sqsubseteq K' \in C$, $c(A_1, \ldots, A_n) \in K$ and $c(B_1, \ldots, B_n) \in K'$, then $A_i \sqsubseteq B_i \in C$ for each $i \in [1..n]$.
(3) If $A \to B \sqsubseteq A' \to B' \in C$ then $A' \sqsubseteq A \in C$ and $B \sqsubseteq B' \in C$.
(4) If $A \rightarrowtail B \sqsubseteq A' \rightarrowtail B' \in C$ then $A \sqsubseteq A' \in C$ and $B' \sqsubseteq B \in C$.

Definition C.3. We say that a constraint $A \sqsubseteq B$ is said to be *syntactically consistent* just if either:

- At least one of A or B is a type variable
- or, A and B are both sufficiency arrows or are both necessity arrows
- or, B is Ok
- or, A is of shape $\Sigma_{c \in I} c(\vec{A})$, B is of shape $\Sigma_{d \in J} d(\vec{B})$ and $I \subseteq J$

We say that a constraint set C is *consistent* just if it is closed and contains only syntactically consistent constraints. We say that a type environment Γ is *consistent* just if $\mathrm{Con}(\Gamma)$ is consistent.

Definition C.4. A judgement of the constrained type system is said to be a *single-subject (1S)* just if it either has shape:

(I) $C \mid \Gamma,\ M : A \vdash \Delta$, Γ and Δ are consistent variable environments with disjoint subjects
(II) or $C \mid \Gamma \vdash M : A$, with Γ a consistent variable environment

Note, a consequence of the 1S definition is that a 1S judgment has at most non-variable subject – here we will consider top-level identifiers also as (global) variables, which is the $M : A$ above. If a 1S judgment has a non-variable subject then it is either of type (I) or type (II), but cannot be both. Otherwise, if a 1S judgement has two formulas on the left-hand side with the same variable subject, then it must be of type (I), with M being one of the two. Otherwise, a 1S judgement with all distinct variable subjects on the left-hand side can be seen as either type (I) or type (II). 1S variable judgments are important because well-typing and ill-typing are both 1S judgments, and a proof of a 1S judgment only involves 1S judgments.

Lemma C.5. *A proof tree $\mathcal{T}$ of a 1S conclusion contains only 1S judgments at each node.*

Proof. By induction on the height of $\mathcal{T}$. When the tree is of height 0, i.e. it is only a (GVar), (LVar) or (VarK) and the result follows by assumption. Otherwise, suppose the tree has height k and we argue by case analysis on the root:

- If the root is concluded by (Disj) then the judgement has shape $C \mid \Gamma,\ M : A \vdash \Delta$. We consider two cases. If M is a non-variable subject, then Γ and Δ are consistent and disjoint variable

environments. Hence, $\Gamma \vdash M : B$ is also 1S and the result follows from the induction hypothesis. Otherwise, M is a variable. If there are no non-variable subjects in Γ then variable M may also occur in Γ – it can be of type (I). Then $\Gamma \vdash M : B$ is 1S. Otherwise M cannot occur as a subject of Γ (which would violate the strong consistency requirement) and hence $\Gamma \vdash M : B$ is also 1S.

- If the root is concluded by (SubL) or (SubR), then clearly the judgement in the premise is also 1S and the result follows from the induction hypothesis.
- If the root is concluded by (AbsR) or (AbnR), then the 1S conclusion has shape $\Gamma \vdash \lambda x.\, M : A$. Thus Γ is a consistent variable environment, and we assume by the side condition that the bound variable x does not occur in Γ. Thus the premises can be seen as 1S type (II) and type (I) respectively, and the result follows from the induction hypothesis.
- If the root is concluded by (AppL) or (AppR), then the 1S conclusion has shape $\Gamma,\, M\,N : A \vdash \Delta$ or $\Gamma \vdash M\,N : A$ respectively. In the former case, Γ and Δ are consistent and disjoint variable environments, and therefore $\Gamma \vdash M : B \rightarrowtail A$ is of type (II) and $\Gamma,\, N : B \vdash \Delta$ is of type (I). In the latter case, Γ is a consistent variable environment and hence $\Gamma \vdash M : B \to A$ and $\Gamma \vdash N : B$ are both of type (II). In both cases, the result follows from the induction hypothesis.
- If the root is concluded by (CnsL), then the 1S conclusion has shape $\Gamma,\, C(M_1, \ldots, M_m) : C(A_1, \ldots, A_m) \vdash \Delta$ and therefore Γ and Δ are disjoint, consistent variable environments. Hence, $\Gamma,\, M_i : A_i \vdash \Delta$ is a 1S judgment of type (I) and the result follows from the induction hypothesis.
- If the root is concluded by (CnsR), then the 1S conclusion has shape $\Gamma \vdash c(M_1, \ldots, M_m) : c(A_1, \ldots, A_m)$ and Γ is a consistent variable environment. Then $\Gamma \vdash M_i : A_i$ is of type (II) for each i and the result follows from the induction hypothesis.
- If the root is concluded by (MchL), then the 1S conclusion is of shape $\Gamma,\, \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\} : A \vdash \Delta$ and so Γ and Δ are consistent, disjoint variable environments. We may assume by the implicit side condition that the variables bound in the patterns are fresh for the context. Hence, it follows immediately that each $\Gamma,\, P_i : A \vdash x : B_x$ is type (I). Similarly, $\Gamma,\, (M, P_i) : (p_i[B_x/x \mid x \in \mathsf{FV}(p_i)], A) \vdash \Delta$ is type (I). Hence, the result follows from the induction hypothesis.
- If the root is concluded by (MchR), then the 1S conclusion is of shape $\Gamma \vdash \mathsf{match}\, M\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\} : A$ and Γ is a consistent variable environment. Moreover, we may assume by the implicit side condition that the variables bound by the patterns are fresh for the context. Therefore, both of the premises are type (II) and the result follows from the induction hypothesis.
- If the root is concluded by (FixR), then the 1S conclusion is of shape $\Gamma \vdash \mathsf{fix}\, x.\, M : A$ and Γ is a consistent variable environment. Therefore, the premise is type (II) and the result follows from the induction hypothesis.
- If the root is concluded by (CnsK), then the 1S conclusion is of shape $\Gamma,\, c(M_1, \ldots, M_n) : \mathsf{Ok} \vdash \Delta$ and Γ and Δ are disjoint, consistent variable environments. Therefore, the premise is type (I) and the result follows from the induction hypothesis.
- If the root is concluded by (FunK), then the 1S conclusion has shape $\Gamma,\, M\,N : \mathsf{Ok} \vdash \Delta$ and Γ and Δ are disjoint, consistent variable environments. Therefore, the premise is type (I) and the result follows from the induction hypothesis.

□

From now on we will consider only 1S judgements.

Lemma C.6 (Left Inversion). *Suppose $\Gamma,\, M : A \vdash \Delta$ has Γ and Δ consistent, disjoint variable environments. Then either (a) Δ is of shape $x : \mathsf{Ok}$, or (b) one of the following is true:*

- *M is of shape x and there is some B such that $x : B = \Delta$ and $\Gamma \vdash A \sqsubseteq B$.*
- *M is of shape f and there is some type B such that $x : B = \Delta$ and $\Gamma \vdash A \sqsubseteq B$*
- *M is of shape PQ and either:*
  - *(i) there is a type B such that $\Gamma \vdash M : B \rightarrowtail A$ and $\Gamma,\ N : B \vdash \Delta$*
  - *(ii) or, $\Gamma,\ M : \mathsf{Ok} \rightarrowtail A \vdash \Delta$.*
- *M is of shape $c(M_1, \ldots, M_m)$ and either:*
  - *(i) there are types $B_1, \ldots, B_m$ and i such that $A \sqsubseteq c(B_1, \ldots, B_m)$ and $\Gamma,\ M_i : B_i \vdash \Delta$.*
  - *(ii) or, $\Gamma,\ M_i : \mathsf{Ok} \vdash \Delta$ for some i*
  - *(iii) or, there is some type A' which is either an arrow or sum type not including a c-headed case, such that $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma,\ c(M_1, \ldots, M_n) : A' \vdash \Delta$*
- *M is of shape $\mathsf{match}\ Q\ \mathsf{with}\ \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$ and there is a family of types B_x (one for each pattern-bound variable x) such that $\Gamma,\ P_i : A \vdash x : B_x$ (for each i and each $x \in \mathsf{FV}(p_i)$), and either $\Gamma,\ Q : p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \vdash \Delta$ or $\Gamma,\ P_i : A \vdash \Delta$.*
- *M is of shape $\lambda x.P$ and there is some sum type A' such that $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma,\ \lambda x.P : A' \vdash \Delta$*

Proof. The proof is by cases on the shape of M.

- If M is a variable x then, according to the assumptions, all the formulas have variables as subjects. By the completeness of algorithmic type assignment, we have $\mathsf{Typ}(\Gamma),\ x : A \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$. Given that Γ and Δ are disjoint, the only rules that can have concluded such a judgement are (VarK2), (LVar2). The former gives rise to (a). In case (LVar2), since Γ and Δ are disjoint, it must be that there is some B such that $x : B \in \Delta$ and $\Gamma \vdash A \sqsubseteq B$, as required by (c).
- If M is a top-level identifier, then the reasoning is as in the former case, since A is a monotype.
- If M is an application PQ, then we reason as follows. By the completeness of algorithmic type assignment, $\mathsf{Typ}(\Gamma),\ P\,Q : A \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$ and the only rules that could have concluded this judgement are (VarK2), (AppL2) or (FunK2). The former case gives rise to (a). In case (AppL2), it follows that there are types B_1 and B such that $\mathsf{Con}(\Gamma) \vdash B_1 \sqsubseteq B \rightarrowtail A$, $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 M : B_1$ and $\mathsf{Typ}(\Gamma),\ N : B \mid \mathsf{Typ}(\Gamma) \vdash_2 \Delta$. It follows from the soundness of algorithmic type assignment that $\Gamma \vdash M : B_1$ and hence, by (SubR), $\Gamma \vdash M : B \rightarrowtail A$. Similarly, $\Gamma,\ N : B \vdash \Delta$. This is what is required by (i). In the latter case it follows that there is a type B such that $\mathsf{Con}(\Gamma) \vdash_2 \mathsf{Ok} \rightarrowtail A \sqsubseteq B$ and $\mathsf{Typ}(\Gamma),\ M : B \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$. It follows from the soundness of algorithmic type assignment that $\Gamma,\ M : B \vdash \Delta$ and hence, by (SubR), $\Gamma,\ M : \mathsf{Ok} \rightarrowtail A \vdash \Delta$, as required by (ii).
- If M is a constructor term $c(M_1, \ldots, M_n)$, then we reason as follows. By the completeness of algorithmic type assignment, $\mathsf{Typ}(\Gamma),\ c(M_1, \ldots, M_n) : A \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$. Since Γ and Δ are disjoint, the only rules that can conclude such a judgement are (VarK2), (ConsL2), (CnsK2) and (CnsDL). The first possibility give rise to (a). The second implies that there are types $A_1, \ldots, A_n$ and $\mathsf{Con}(\Gamma) \vdash A \sqsubseteq c(A_1, \ldots, A_n)$ and $\mathsf{Typ}(\Gamma),\ M_i : A_i \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$ for some i. By the soundness of algorithmic type assignment, $\Gamma,\ M_i : A_i \vdash \Delta$, as required by (i). In the third, we have for some i, $\mathsf{Typ}(\Gamma),\ M_i : \mathsf{Ok} \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$. Then (ii) follows by the soundness of algorithmic type assignment. Finally, when the conclusion is by (CnsDL), there is some type A' which is either an arrow or a sum not including c, such that $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma,\ M : A' \vdash \Delta$.
- If M is a match expression $\mathsf{match}\ Q\ \mathsf{with}\ \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$ then we reason as follows. It follows from the completeness of algorithmic type assignment that $\mathsf{Typ}(\Gamma),\ \mathsf{match}\ Q\ \mathsf{with}\ \{|_{i=1}^{k}\ (p_i \mapsto P_i)\} : A \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$. Given the disjointness of the environments, the only rules that could conclude this judgement are (VarK2) and (MchL2). In the former case we obtain (a),

so let us assume that Δ is not of this shape. In the latter case, we have that there are families of types B_i, A_i, A'_i, A_x and B_x, such that (1) $\mathsf{Typ}(\Gamma)$, $(Q, P_i) : A'_i \mid \mathsf{Con}(\Gamma) \vdash_2 \Delta$ for all i, (2) $\mathsf{Typ}(\Gamma)$, $P_i : A_x \mid \mathsf{Con}(\Gamma) \vdash_2 x : B_x$ for all i and all $x \in \mathsf{FV}(p_i)$, $\mathsf{Con}(\Gamma) \vdash A \sqsubseteq A_i$ for all i, $\mathsf{Con}(\Gamma) \vdash A \sqsubseteq A_x$ for all i and all $x \in \mathsf{FV}(p_i)$, $\mathsf{Con}(\Gamma) \vdash p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq B_i$ for all i, and $\mathsf{Con}(\Gamma) \vdash (B_i, A) \sqsubseteq A'_i$ for all i. It follows from the soundness of algorithmic typing that Γ, $(Q, P_i) : A'_i \vdash \Delta$ for all i, and Γ, $P_i : A_x \vdash x : B_x$ for all i and all $x \in \mathsf{FV}(p_i)$. Then it follows from the previous case that either (1) $A'_i = \mathsf{Ok}$ and either Γ, $Q : \mathsf{Ok} \vdash \Delta$ or Γ, $P_i : \mathsf{Ok} \vdash \Delta$, or (2) A'_i is an arrow type or (3) there are types B''_1 and B''_2 such that $\Gamma \vdash A'_i \sqsubseteq (B''_1, B''_2)$ and Γ, $Q : B''_1 \vdash \Delta$ or Γ, $P_i : B''_2 \vdash \Delta$. In case (1), it follows from (SubL) that either Γ, $Q : p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \vdash \Delta$ or Γ, $P_i : A \vdash \Delta$, as required. Case (2) is impossible by the consistency of Γ. In case (3), we have $\Gamma \vdash (p_i[B_x/x \mid x \in \mathsf{FV}(p_i)], A) \sqsubseteq (B_i, A) \sqsubseteq A'_i \sqsubseteq (B''_1, B''_2)$ and thus, by the closedness of Γ, we have $\Gamma \vdash p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \sqsubseteq B''_1$ and $\Gamma \vdash A \sqsubseteq B''_2$. Then it follows from (SubL) that either Γ, $Q : p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \vdash \Delta$ or Γ, $P_i : A \vdash \Delta$

- When M is an abstraction $\lambda x.\, P$, it follows from the completeness of algorithmic type assignment that Γ, $\lambda x.\, P : A \vdash_2 \Delta$ and then the only applicable rules are (VarK) and (AbsDL). The former case gives rise to (a). In the latter case, it is immediate that there is some sum type A' such that $\Gamma \vdash A \sqsubseteq A'$ and Γ, $M : A' \vdash \Delta$.

□

Lemma C.7 (Right Inversion). *Suppose $\Gamma \vdash M : A$ and Γ a consistent variable environment. Then one of the following is true:*

- *M is of shape x and $A = \mathsf{Ok}$ or there is some B such that $x : B \in \Gamma$ and $\Gamma \vdash B \sqsubseteq A$.*
- *M is of shape f and there are types B, $a_1, \ldots, a_n$ and $B_1, \ldots, B_n$, and constraints D such that $f : \forall\, \vec{a}\,.\, D \Rightarrow B \in \Gamma$ and $\Gamma \vdash B[\vec{B}/\vec{a}] \sqsubseteq A$*
- *M is of shape PQ and there is a type B such that $\Gamma \vdash P : B \to A$ and $\Gamma \vdash N : B$*
- *M is of shape $\lambda x.\, P$ and there are types B_1 and B_2 such that either:*
  (i) Γ, $x : B_1 \vdash M : B_2$ and $\Gamma \vdash B_1 \to B_2 \sqsubseteq A$
  (ii) or, Γ, $M : B_2 \vdash x : B_1$ and $\Gamma \vdash B_1 \rightarrowtail B_2 \sqsubseteq A$
- *M is of shape $c(M_1, \ldots, M_n)$ and there are types $B_1, \ldots, B_n$ such that $\Gamma \vdash M_i : B_i$ for each i, and $\Gamma \vdash c(B_1, \ldots, B_n) \sqsubseteq A$.*
- *M is of shape $\mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$ and there is a family of types B_x (for each i and each $x \in \mathsf{FV}(p_i)$) such that $\Gamma \vdash Q : \Sigma_{i=1}^{k} p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$ and $\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A$ (for each i).*
- *M is of shape $\mathsf{fix}\, x.\, P$ and Γ, $x : A \vdash M : A$.*

Proof. By cases on the shape of M.

- When M is a local variable x we reason as follows. By the completeness of algorithmic type assignment, we have $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 x : A$. Given that Γ is consistent, the only rules that can conclude the judgement are (Var2) and (VarK2). In the latter case, $A = \mathsf{Ok}$. In the former case, there is some B such that $x : B \in \mathsf{Typ}(\Gamma)$ and $\mathsf{Con}(\Gamma) \vdash B \sqsubseteq A$.
- When M is an identifier f, by the completeness of type assignment we have that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 f : A$. The only applicable rule was (Inst2) and hence there are type variables $\vec{a}$ and types $\vec{B}$ and B, and constraints D such that $f : \forall\, \vec{a}\,.\, D \Rightarrow B \in \mathsf{Typ}(\Gamma)$ and $\mathsf{Con}(\Gamma) \vdash D[\vec{B}/\vec{a}]$ and $\mathsf{Con}(\Gamma) \vdash B[\vec{B}/\vec{a}] \sqsubseteq A$.
- When M is an application PQ, by the completeness of type assignment, we have that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 PQ : A$. The only applicable rule is (AppR2) and so there are types B_1 and B such that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 P : B_1$, $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 Q : B$ and $\mathsf{Con}(\Gamma) \vdash B_1 \sqsubseteq B \to A$. It

follows from the soundness of algorithmic type assignment that $\Gamma \vdash P : B_1$ and $\Gamma \vdash Q : B$ and by (()SubR) also $\Gamma \vdash P : B \to A$, as required.

- When M is an abstraction $\lambda x.P$, by the completeness of algorithmic type assignment, we have that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 \lambda x.P : A$. Only the rules (AbsR2) and (AbnR2) could have concluded. In the former case, we have that there are types B_1 and B_2 such that $\mathsf{Typ}(\Gamma), x : B_1 \mid \mathsf{Con}(\Gamma) \vdash M : B_2$ and $\mathsf{Con}(\Gamma) \vdash B_1 \to B_2 \sqsubseteq A$, as required by (i). The latter case is analogous.
- When M is a constructor term $c(M_1, \ldots, M_n)$, by the completeness of algorithmic type assignment, we have that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 c(M_1, \ldots, M_n) : A$. The only rule that could conclude this is (ConsR2), and hence there must be types $B_1, \ldots, B_n$ such that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 M_i : B_i$ for each i and $\mathsf{Con}(\Gamma) \vdash c(B_1, \ldots, B_n) \sqsubseteq A$. It follows from the soundness of algorithmic type assignment that $\Gamma \vdash M_i : B_i$ for each i, as required.
- When M is of shape $\mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$, it follows from the completeness of algorithmic type assignment that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash \mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\} : A$. The only applicable rule is (MchR) and so there is a type B and families of types A_i (for each i) and B_x (for each i and each $x \in \mathsf{FV}(p_i)$) such that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 Q : B$ and $\mathsf{Typ}(\Gamma) \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \mid \mathsf{Con}(\Gamma) \vdash_2 P_i : A_i$ (for each i) and $\mathsf{Con}(\Gamma) \vdash B \sqsubseteq \Sigma_{i=1}^{k} p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$ and $\mathsf{Con}(\Gamma) \vdash A_i \sqsubseteq A$ (for all i). It follows from the soundness of algorithmic type assignment that $\Gamma \vdash Q : B$ and, by (SubR), $\Gamma \vdash \Sigma_{i=1}^{k} p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$. Similarly, $\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A_i$ and, by (SubR), $\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A$.
- When M is of shape $\mathsf{fix}\, x.P$, it follows from the completeness of algorithmic type assignment that $\mathsf{Typ}(\Gamma) \mid \mathsf{Con}(\Gamma) \vdash_2 \mathsf{fix}\, x.P : A$. The only rule that could conclude this judgement is (FixR2) and so there is some type B such that $\mathsf{Typ}(\Gamma), x : A \mid \mathsf{Con}(\Gamma) \vdash P : B$ and $\mathsf{Con}(\Gamma) \vdash B \sqsubseteq A$. Hence, it follows from soundness of algorithmic type assignment and (SubR) that $\Gamma, x : A \vdash M : A$, as required.

□

Lemma C.8. *If $C \vdash A_0 \sqsubseteq A_n$ then there is a sequence of constraints:*

$$A_1 \sqsubseteq A_2, A_3 \sqsubseteq A_4, \ldots A_{n-2} \sqsubseteq A_{n-1}$$

all of which are in C and such that:

(i) for all $i \in [0..n/2]$ either A_{2i+1} is Ok *or otherwise:*
  (a) A_{2i} is a type variable and $A_i = A_{2i+1}$
  (b) or, A_{2i} is a sufficiency arrow and so is A_{2i+1}
  (c) or, A_{2i} is a necessity arrow and so is A_{2i+1}
  (d) or, A_{2i} is of shape $\Sigma_{c \in I}\, c(\vec{A})$ and A_{2i+1} is of shape $\Sigma_{d \in J}\, d(\vec{A})$ and $I \subseteq J$.

Proof. The proof is by induction on the derivation of $C \vdash A \sqsubseteq B$.

(IdS) Clearly this satisfies the conclusion with a sequence of length 1.

(TrS) It follows from the induction hypotheses that there are sequences

$$A_1 \sqsubseteq A_2, A_3 \sqsubseteq A_4, \ldots A_{n-2} \sqsubseteq A_{n-1} \qquad \text{and} \qquad B_1 \sqsubseteq B_2, B_3 \sqsubseteq B_4, \ldots B_{m-2} \sqsubseteq B_{m-1}$$

satisfying the appropriate properties. The middle witness is $A_n = B_0$. If it is a type variable, then $A_{n-1} = A_n = B_0 = B_1$. If it is an arrow, then A_{n-1} and B_1 are arrows of the same kind too. If it is a sum $\Sigma_{d \in J}\, d(\ldots)$, then A_{n-1} is a sum $\Sigma_{c \in I}\, c(\ldots)$ with $I \subseteq J$ and B_1 is a sum $\Sigma_{k \in K}\, k(\ldots)$ with $J \subseteq K$. If it is Ok, then there is no requirement on A_{n-1} but B_1 must be Ok. Hence, in each case, A_{n-1} and B_1 satisfy the required relationship and thus the witness is just the concatenation of the sequences.

(ToS),(FrS),(SmS),(CnS),(OkS) The conclusion is obtained using a sequence of length 0 because A_0 and A_1 already satisfy the required relationship.

□

Consistent constraint sets do not derive inconsistencies.

Lemma C.9. *Suppose C is syntactically consistent.*

- *If $C \vdash A \sqsubseteq B$ then $A \sqsubseteq B$ is syntactically consistent.*
- *If $C \vdash A \mid\mid B$ then $A \mid\mid B$ is syntactically consistent.*

Proof.
- If $C \vdash A \sqsubseteq B$, then by the forgoing lemma there is a sequence of inequalities in C satisfying the required conditions:

$$A_1 \sqsubseteq A_2, A_3 \sqsubseteq A_4, \ldots A_{n-2} \sqsubseteq A_{n-1}$$

  Suppose for the purpose of obtaining a contradiction that $A \sqsubseteq B$ is not syntactically consistent. Then there is a consecutive subsequence of the above, of shape:

$$B_i \sqsubseteq a_1, a_1 \sqsubseteq a_2, a_2 \sqsubseteq a_3 \ldots, a_k \sqsubseteq B_{2k+1}$$

  (i.e. intermediate types are all type variables) of constraints in C in which $B_i \sqsubseteq B_{2k+1}$ is already not syntactically consistent. However, it follows that $B_i \sqsubseteq B_{2k+1}$ is contained in the closure of C, which is assumed consistent.
- The proof is by induction on the derivation of $C \vdash A \mid\mid B$. In cases (ConD), (ToD) and (FromD) the result is immediate. In case (Rfld) the result follows from the assumption. In case (SymD) the result follows from the induction hypothesis since the characterisation of syntactic consistency is symmetrical. Finally, in case (SubD) we can extend Lemma C.8 to show that there are two sequences of subtype constraints in C each of which gives rise to one side of the disjointness constraint. Then it follows similarly, that if a disjointness constraint is concluded that is not syntactically consistent, then already there would be a non-syntactically consistent constraint in the closure.

□

Lemma C.10. *Suppose $\mathsf{Con}(\Gamma)$ is closed and syntactically consistent. If $\Gamma \vdash V : A$ with V a closed value and A not a type variable, then either:*

- *A is a sum type containing a type of shape $c(A_1, \ldots, A_n)$ and V a constructor term of shape $c(W_1, \ldots, W_n)$ and $\Gamma \vdash W_i : A_i$ for each i,*
- *or, A is an arrow and V is an abstraction*

Proof. We proceed by cases on the shape of V.

- If V is of shape $c(W_1, \ldots, W_n)$ then, by inversion, there are types $B_1, \ldots, B_n$ such that $\Gamma \vdash M_i : B_i$ for each i, and $\Gamma \vdash c(B_1, \ldots, B_n) \sqsubseteq A$. Therefore, it follows from consistency of subtyping (Lemma C.9) that either A is a sum type including a c-headed type or A is a type variable.
- If V is of shape $\lambda x.M$ then, by inversion, there are types B_1 and B_2 such that either $\Gamma \vdash B_1 \to B_2 \sqsubseteq A$ or $\Gamma \vdash B_1 \rightarrowtail B_2 \sqsubseteq A$. By consistency of subtyping, in both cases, A is either a type variable or an arrow.

□

C.0.1 *Proof of Theorem 5.10 (Progress).*

Proof. The first is by induction on the derivation. Due to the form of the judgement, every subject in Γ can make a step, so we focus on the principal formula in each derivation step and exclude cases that cannot occur.

(GVar) in this case M is a top-level identifier, so M can make a step.

(SubR) It follows from the induction hypothesis that M can make a step, or M is a value.

(AbsR) It is immediate that M is a value.

(AbnR) It is immediate that M is a value.

(AppR) In this case, M has shape PQ and it follows from the induction hypothesis that either P can make a step or is a value and Q can make a step or is a value. If P can make a step, then M can make a step, so assume to the contrary. Then P is a value. It follows that from Lemma C.10 that P must be an abstraction. Now, if Q can make a step, then M can make a step and if, not, then it is a value and M can make a step by contracting the redex.

(CnsR) In this case, M is of shape $c(P_1, \ldots, P_n)$ and it follows from the induction hypothesis that, for each i, P_i can either make a step or is a value. From any configuration it follows that M can either make a step or is a value.

(MchR) In this case, M is of shape $\mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$. It follows from the induction hypothesis that Q can either make a step or is a value. If Q can make a step, then M can make a step. Otherwise, Q is a value of a sum type and it follows from Lemma C.10 that Q must be a constructor term $c(W_1, \ldots, W_n)$ and there is a c-headed type in the sum. Hence, there is a pattern p_i of shape $c(x_1, \ldots, x_n)$ and so M can make a step.

(FixR) It is immediate that M can make a step.

The second is also by induction on the derivation. Similar remarks apply. Note that, if M is stuck, then it follows from the previous result that $\Gamma \nvdash M : A$ for any A.

(Disj) It follows from the previous result that M can either make a step or is a value (of some type B disjoint from A). In the former case, the result is immediate. In the latter case, if M is of shape $c(W_1, \ldots, W_n)$ then it follows from inversion that there are types $A_1, \ldots, A_n$ such that $\Gamma \vdash c(A_1, \ldots, A_n) \sqsubseteq B$. Then it follows from subtype consistency that $\Gamma \nvdash c(B_1, \ldots, B_n) \sqsubseteq A$ for any types $B_1, \ldots, B_n$. Therefore, it follows from inversion that $\Gamma \nvdash M : A$.

(SubL) In this case we assume $\Gamma \vdash A \sqsubseteq B$. It follows from the induction hypothesis that either M makes a step or $\Gamma \nvdash M : B$. In the former case the result is immediate, in the latter case, we must have $\Gamma \nvdash M : A$ too by the contrapositive of (SubR).

(AppL) In this case, M is of shape PQ. Since M is not a value, either M can make a step or M is stuck and hence the desired conclusion follows in both cases.

(CnsL) In this case, M is of shape $c(M_1, \ldots, M_n)$ and A is of shape $c(A_1, \ldots, A_n)$. It follows from the induction hypothesis that either M_i can make a step or $\Gamma \nvdash M_i : A_i$. In the former case, either M can make a step, or M is stuck. In either case, the result follows. If $\Gamma \nvdash M_i : A_i$ then we have that $\Gamma \nvdash M : A$. To see this, suppose the contrary. Then it follows from inversion that there are types $B_1, \ldots, B_n$ such that $\Gamma \vdash M_i : B_i$ and $\Gamma \vdash c(B_1, \ldots, B_n) \sqsubseteq c(A_1, \ldots, A_n)$. Since $\mathsf{Con}(\Gamma)$ is closed, it follows that $\Gamma \vdash B_i \sqsubseteq A_i$ and so $\Gamma \vdash M_i : A_i$ follows by (SubR), which contradicts our supposition.

(MchL) In this case, M is of shape $\mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$. Since M is not a value, either M can make a step or is stuck. The desired conclusion follows in both cases.

(CnsK) In this case, M is of shape $c(M_1, \ldots, M_n)$ and A is Ok. It follows from the induction hypothesis that there is some M_i that can either make a step or for which $\Gamma \nvdash M_i : \mathsf{Ok}$. In the former

case, either M can make a step or M is stuck and the result follows. In the latter case, since all values are well-typed, it follows that M_i is stuck and so the result follows.

(FunK) In this case, M is of shape $M\,N$ and A is Ok. Since M is not a value, it can either make a step or is stuck and in both cases the desired conclusion follows.

□

Lemma C.11 (Substitution on the right). *Assume Γ is a consistent variable environment, disjoint from Δ and that does not contain x.*

(i) If $\Gamma,\, x : B \vdash M : A$ and $\Gamma \vdash N : B$, then $\Gamma \vdash M[N/x] : A$.
(ii) If $\Gamma,\, x : B,\, M : A \vdash \Delta$ and $\Gamma \vdash N : B$, then $\Gamma,\, M[N/x] : A \vdash \Delta$.

Proof. The proof is by induction on M. Note that, in (ii), it is always possible that Δ is of shape $z : \mathrm{Ok}$ for some variable z. Then the conclusion follows immediately by (VarK), so we exclude this from the case analysis below.

- When M is x, we reason as follows. In (i), it follows by inversion that $\Gamma \vdash B \sqsubseteq A$. Since $M[N/x] = N$, we have $\Gamma \vdash M[N/x] : B$ and by (SubR), therefore $\Gamma \vdash M[N/x] : A$. In (ii) it follows from inversion that Δ must contain $y : \mathrm{Ok}$ for some variable y, which we have dealt with above.
- When M is some variable y distinct from x, we reason as follows. In (i), it follows from the assumptions and inversion that $y : A' \in \Gamma$ with $\Gamma \vdash A' \sqsubseteq A$. Since $M[N/x] = M = y$, we have by (LVar) that $\Gamma \vdash M[N/x] : A'$ and the conclusion follows from (SubR). In (ii), Δ is $y : A'$ with $A \sqsubseteq A'$. Then the conclusion follows from (LVar) and (SubL).
- When M is an application $P\,Q$, we reason as follows. In (i), by inversion we have that there is some B' such that $\Gamma,\, x : B \vdash P : B' \to A$ and $\Gamma,\, x : B \vdash Q : B'$. It follows from the induction hypotheses that $\Gamma \vdash P[N/x] : B' \to A$ and $\Gamma \vdash Q[N/x] : B'$ and so the conclusion follows from (AppR). In (ii), inversion gives us that either: (1) there is a type B' and $\Gamma,\, x : B \vdash P : B' \rightarrowtail A$ and $\Gamma,\, x : B,\, Q : B' \vdash \Delta$, or (2) $\Gamma,\, x : B,\, P : \mathrm{Ok} \rightarrowtail A \vdash \Delta$. Suppose the former, then it follows from the induction hypothesis part (i) that $\Gamma \vdash P[N/x] : B' \rightarrowtail A$ and from part (ii) that $\Gamma,\, Q[N/x] : B' \vdash \Delta$. Hence the conclusion follows from (AppL). Suppose the latter, then it follows from the induction hypothesis part (ii) that $\Gamma,\, P[N/x] : \mathrm{Ok} \rightarrowtail A$, and the conclusion follows from (FunK).
- When M is an abstraction $\lambda y.\, P$, we may assume by the variable convention that y does not occur outside of P. In (i), by inversion we have that there are types B_1 and B_2 such that either (1) $\Gamma,\, x : B,\, y : B_1 \vdash P : B_2$ and $\Gamma \vdash B_1 \to B_2 \sqsubseteq A$ or (2) $\Gamma,\, x : B,\, P : B_2 \vdash y : B_1$ and $\Gamma \vdash B_1 \rightarrowtail B_2 \sqsubseteq A$. In case (1), it follows from the induction hypothesis part (i) that $\Gamma,\, y : B_1 \vdash P[N/x] : B_2$ and thus the conclusion follows from (AbsR) and (SubR). In case (2) it follows from the induction hypothesis part (ii) that $\Gamma,\, P[N/x] : B_2 \vdash y : B_1$ and hence the conclusion follows from (AbnR) and (SubR). In (ii), by inversion we have that there is some arrow type or sum type A' and $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma,\, x : B,\, \lambda y.\, P : A' \vdash \Delta$. In either case, the conclusion follows from (AbsDL) and (SubL).
- When M is a constructor term $c(P_1, \ldots, P_n)$, we reason as follows. In (i) it follows from inversion that there are types $B_1, \ldots, B_n$ and $\Gamma,\, x : B \vdash P_i : B_i$ for each i and $\Gamma \vdash c(B_1, \ldots, B_n) \sqsubseteq A$. It follows from the induction hypothesis that $\Gamma \vdash c(P_1[N/x], \ldots, P_n[N/x]) : c(B_1, \ldots, B_n)$ and then the conclusion follows from (SubR). In (ii), it follows from inversion that either (1) there are types $B_1, \ldots, B_n$ and i such that $\Gamma \vdash A \sqsubseteq c(B_1, \ldots, B_n)$ and $\Gamma,\, x : B,\, P_i : B_i \vdash \Delta$, or (2) there is an arrow type A' such that $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma,\, x : B,\, c(P_1, \ldots, P_n) : A' \vdash \Delta$. In the former case, the conclusion follows from the induction hypothesis and (SubL), in the latter it follows from (CnsDL) and (SubL).

- When M is a match expression $\mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$, we reason as follows. In (i), by inversion we have a family of types B_x (for each i and $x \in \mathsf{FV}(p_i)$) such that $\Gamma, x : B \vdash Q : \Sigma_{i=1}^{k} p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$ and $\Gamma, x : B \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A$ for each i. The conclusion follows from the induction hypothesis, part (i). In (ii), by inversion we have a family of types B_y (for each pattern-bound variable y) such that $\Gamma, x : B, P_i : A \vdash y : B_y$ and either $\Gamma, Q : p_i[B_y/y \mid y \in \mathsf{FV}(p_i)] \vdash \Delta$ or $\Gamma, P_i : A \vdash \Delta$. In both cases, it follows from the induction hypothesis and (CnsL) that $\Gamma, (Q[N/x], P_i[N/x]) : (p_i[B_y/y \mid y \in \mathsf{FV}(p_i)], A) \vdash \Delta$. Then we can obtain the conclusion by applying the induction hypothesis part (ii) to each of the former judgements and concluding by (MchL).
- When M is a fixpoint expression $\mathsf{fix}\, y.\, P$, we reason as follows. In (i), it follows from inversion that $\Gamma, x : B, y : A \vdash P : A$ and the result follows from the induction hypothesis part (i) and (FixR). In (ii), it follows from inversion that Δ must have shape $z : \mathsf{Ok}$, which we have covered in the above observation.

□

Lemma C.12 (Trivial Substitution). *Assume Γ is a consistent variable environment disjoint from Δ and neither contain x. Assume $\Gamma \vdash V : B$ is a closed value. Then:*

(i) if $\Gamma \vdash M : A$ then $\Gamma \vdash M[V/x] : A$
(ii) if $\Gamma, M : A \vdash \Delta$ then $\Gamma, M[V/x] : A \vdash \Delta$

Proof. The proof is by induction on M. Note, in case (ii), there is always the possibility that Δ has shape $z : \mathsf{Ok}$ for some variable z and then the conclusion follows immediately from (VarK). Moreover, if M does not contain x, then the conclusion is immediate. Hence, we will exclude these cases from consideration below.

- If M is x, we reason as follows. In (i), by inversion, it must be that $A = \mathsf{Ok}$ and the conclusion follows because all closed values are well-typed. In (ii), by inversion, it must be that Δ is of shape $z : \mathsf{Ok}$ as above.
- If M is of shape PQ we reason as follows. In (i) by inversion it must be that there is some type B such that $\Gamma \vdash P : B \to A$ and $\Gamma \vdash Q : B$. The conclusion follows from the induction hypotheses, part (i). In (ii) by inversion it must be that either (1) there is a type B such that $\Gamma \vdash P : B \rightarrowtail A$ and $\Gamma, Q : B \vdash \Delta$, or (2) $\Gamma, P : \mathsf{Ok} \rightarrowtail A \vdash \Delta$. In the former case, the result follows from the induction hypotheses and (AppL). In the latter case, the result follows from the induction hypothesis and (FunK).
- If M is of shape $c(P_1, \ldots, P_n)$ we reason as follows. In (i), by inversion, it must be that there are types $B_1, \ldots, B_n$ and $\Gamma \vdash c(B_1, \ldots, B_n) \sqsubseteq A$ and $\Gamma \vdash P_i : B_i$ for each i. The result follows from the induction hypothesis, (CnsR) and (SubR). In (ii), by inversion, it follows that either (1) there are types $B_1, \ldots, B_n$ and i such that $\Gamma \vdash A \sqsubseteq c(B_1, \ldots, B_n)$ and $\Gamma, P_i : B_i \vdash \Delta$, or (2) there is an arrow type A' such that $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma, c(P_1, \ldots, P_n) : A' \vdash \Delta$. In the former case, the result follows from the induction hypotheses, (CnsL) and (SubL). In the latter case, the result follows from (CnsDL).
- If M is of shape $\lambda y.\, P$, we may assume that y does not occur outside of P. In (i), by inversion, it must be that there are types B_1 and B_2 such that, either: (1) $\Gamma, y : B_1 \vdash P : B_2$ and $\Gamma \vdash B_1 \to B_2 \sqsubseteq A$ or (2) $\Gamma, P : B_2 \vdash y : B_1$ and $\Gamma \vdash B_1 \rightarrowtail B_2 \sqsubseteq A$. In (1), the result follows from the induction hypothesis, (AbsR) and (SubR). In (2), the result follows from the induction hypothesis, (AbnR) and (SubR). In (ii), by inversion, it must be that there is some sum type A' such that $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma \vdash \lambda y.\, P : A' \vdash \Delta$. Then the result follows from (AbsDL).
- If M is of shape $\mathsf{fix}\, y.\, P$ then we may assume that y occurs nowhere else. In (i), by inversion we have that $\Gamma, y : A \vdash P : A$ and the result follows from the induction hypothesis and

(FixR). In (ii), by inversion the only possibility is that Δ is of shape $z : \mathsf{Ok}$, which is explained above.

- If M is of shape $\mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$, we reason as follows. In (i), it follows from inversion that there is a family of types B_y (for each pattern-bound variable y) and $\Gamma \vdash Q : \Sigma_{i=1}^{k} p_i[B_y/y \mid x \in \mathsf{FV}(p_i)]$ and $\Gamma \cup \{y : B_y \mid y \in \mathsf{FV}(p_i)\} \vdash P_i : A$ for each i. Then the conclusion follows from the induction hypothesis and (MchR). In (ii), by inversion we have a family of types B_y (one for each pattern-bound variable y) such that $\Gamma, P_i : A \vdash y : B_y$ (for each i and each $y \in \mathsf{FV}(p_i)$) and either $\Gamma, Q : p_i[B_y/y \mid y \in \mathsf{FV}(p_i)] \vdash \Delta$ or $\Gamma, P_i : A \vdash \Delta$. In both cases, it follows from the induction hypothesis and (CnsL) that $\Gamma, (Q[V/x], P_i[V/x]) : (p_i[B_y/y \mid y \in \mathsf{FV}(p_i)], A) \vdash \Delta$. Then the conclusion follows from the induction hypothesis and (MchL).

□

Lemma C.13 (Substitution on the Left). *Assume Γ is a consistent identifier environment disjoint from Δ and neither contains x. If $\Gamma, M : A \vdash x : B$ and $\Gamma, V : B \vdash \Delta$ then $\Gamma, M[V/x] : A \vdash \Delta$.*

Proof. The proof is by induction on M. Note, there is always the possibility that $B = \mathsf{Ok}$, in which case we have $\Gamma, V : \mathsf{Ok} \vdash \Delta$ for closed value V and so it follows from inversion that Δ must be of shape $z : \mathsf{Ok}$ for some variable z. Hence, the desired conclusion follows in this case immediately from (VarK).

- If M is x, then it follows from inversion that $\Gamma \vdash A \sqsubseteq B$. Hence, the result follows from the assumed $\Gamma, V : B \vdash \Delta$ and (SubL), since $M[V/x] = V$.
- If M is f, then it follows from inversion that B must be Ok, and we have proven this case above.
- If M is a variable y distinct from x, the reasoning is as in the previous case.
- If M is an application PQ, then it follows from inversion that either (i) there is a type B' and $\Gamma \vdash P : B' \rightarrowtail A$ and $\Gamma, Q : B' \vdash x : B$, or (ii) $\Gamma, P : \mathsf{Ok} \rightarrowtail A \vdash x : B$. In (i), it follows from Lemma C.12 that $\Gamma \vdash P[V/x] : B' \rightarrowtail A$ and it follows from the induction hypothesis that $\Gamma, Q[V/x] : B' \vdash \Delta$. Therefore, the conclusion follows from (AppL). In (ii), the result follows from the induction hypothesis and (FunK).
- If M is an abstraction $\lambda y.\, P$ we may assume y is fresh for the context. Then it follows from inversion that there is some sum type A' such that $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma, \lambda y.\, P : A' \vdash x : B$. Then the result follows from (AbsDL).
- If M is a fixpoint $\mathsf{fix}\, y.\, P$, we may assume that y is fresh for the context. By inversion, it can only be that $B = \mathsf{Ok}$, which we have covered above.
- If M is a constructor $c(P_1, \ldots, P_n)$ then it follows from inversion that either (1) there are types $B_1, \ldots, B_n$ and $\Gamma \vdash A \sqsubseteq c(B_1, \ldots, B_n)$ and $\Gamma, P_i : B_i \vdash x : B$ for some i, or (2) there is some arrow type A' such that $\Gamma \vdash A \sqsubseteq A'$ and $\Gamma, c(P_1, \ldots, P_n) : A' \vdash x : B$. In case (1), the result follows from the induction hypothesis and (CnsL) and (SubL). In case (2), the result follows immediately from (CnsDL) and (SubL).
- If M is a match expression $\mathsf{match}\, Q\, \mathsf{with}\, \{|_{i=1}^{k}\ (p_i \mapsto P_i)\}$ then it follows from inversion that there is some family B_y (for each pattern-bound variable y) and $\Gamma, P_i : A \vdash y : B_y$ (for each i and $y \in \mathsf{FV}(p_i)$) and either $\Gamma, Q : p_i[B_y/y \mid y \in \mathsf{FV}(p_i)] \vdash x : B$ or $\Gamma, P_i : A \vdash x : B$. In both cases, it follows from the induction hypothesis and (CnsL) that $\Gamma, (Q[V/x], P_i[V/x]) : (p_i[B_y/y \mid y \in \mathsf{FV}(p_i)], A) \vdash x : B$. Then the result follows from the induction hypothesis and (MchL).

□

Theorem C.14 (Preservation on the Right). *Suppose* Γ *and* Δ *are disjoint, consistent variable environments with* $\vdash \mathcal{M} : \Gamma$. *If* $\Gamma \vdash M : A$ *and* $M \rhd N$ *then* $\Gamma \vdash N : A$.

Proof. We prove the result for all $\mathcal{E}$, redexes P and contractions Q such that $M = \mathcal{E}[P] \rhd \mathcal{E}[Q] = N$, by induction on $\mathcal{E}$.

- If the context is just a hole, then we proceed by cases on the redex.
  - **(Beta)** In this case, P has shape $(\lambda x.\, P')\, V$ and $Q = P'[V/x]$. It follows from inversion that there is a type B such that $\Gamma \vdash \lambda x.\, P' : B \to A$ and $\Gamma \vdash V : B$. By inversion and the consistency of Γ, we have some B_1 and B_2 such that $\Gamma, x : B_1 \vdash P' : B_2$ and $\Gamma \vdash B_1 \to B_2 \sqsubseteq B \to A$ (the other possibility would involve a type constraint that is not syntactically consistent). By the closedness of the constraints in Γ, we have $\Gamma \vdash B_2 \sqsubseteq A$. Then it follows from the substitution lemma that $\Gamma \vdash P'[V/x] : B_2$ and hence $P'[V/x] : A$ by (SubR).
  - **(Delta)** In this case, it follows from inversion that there is some type B and family $\vec{B}$ such that $f : \forall\, \vec{a}\,.\, D \Rightarrow B$ and $\Gamma \vdash B[\overrightarrow{B/a}] \sqsubseteq A$ and $\Gamma \vdash D[\overrightarrow{B/a}]$. Then it follows from $\vdash \mathcal{M} : \Gamma$ that $\Gamma \cup D \vdash M : B$ and therefore $\Gamma[\overrightarrow{B/a}] \cup D[\overrightarrow{B/a}] \vdash M : B[\overrightarrow{B/a}]$. However, Γ is guaranteed to be closed with respect to type variables, and so we have $\Gamma \cup D[\overrightarrow{B/a}] \vdash M : B[\overrightarrow{B/a}]$. Since also $\Gamma \vdash D[\overrightarrow{B/a}]$, it follows that $\Gamma \vdash M : B[\overrightarrow{B/a}]$, as required.
  - **(Fix)** In this case, P has shape $\mathsf{fix}\, x.\, P'$ and Q is $P'[\mathsf{fix}\, x.\, P'/x]$. It follows from inversion that $\Gamma, x : A \vdash P' : A$. Hence, by substitution on the right, $\Gamma \vdash P'[\mathsf{fix}\, x.\, P'/x] : A$.
  - **(Match)** In this case, P has shape $\mathsf{match}\, c(V_1, \ldots, V_n)\, \mathsf{with}\, \{|_{i=1}^{n} (p_i \mapsto P_i)\}$ and one of the alternatives is of shape $c(x_1, \ldots, x_n) \mapsto P$ and $Q = P[V_i/x_i \mid 1 \le i \le n]$ It follows from inversion that there is a family of types B_x (one for each pattern-bound variable x) such that $\Gamma \vdash c(V_1, \ldots, V_n) : \Sigma_{i=1}^{k} p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$ and $\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P : A$. Therefore, it follows by value inversion that $\Gamma \vdash V_i : B_i$. Moreover, it follows by induction on n and the substitution lemma (since the x_i are disjoint and the V_i closed) that $\Gamma \vdash P[V_i/x_i \mid 1 \le i \le n] : A$, as required.
- If the context is of shape $\mathcal{E}\, M'$ then it follows by inversion that there is some B such that $\Gamma \vdash \mathcal{E}[P] : B \to A$ and $\Gamma \vdash M' : B$. Then it follows from the induction hypothesis that $\Gamma \vdash \mathcal{E}[Q] : B \to A$ hence the result follows by (AppR).
- If the context is of shape $(\lambda x.\, M')\, \mathcal{E}$ then it follows by inversion that there is some B such that $\Gamma \vdash \lambda x.\, M' : B \to A$ and $\Gamma \vdash \mathcal{E}[P] : B$. Then the result follows from the induction hypothesis and (AppR).
- If the context is of shape $\mathsf{match}\, \mathcal{E}\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\}$ then it follows from inversion that there is some family B_x and $\Gamma \vdash \mathcal{E}[P] : \Sigma_{i=1}^{k} p_i[B_x/x \mid x \in \mathsf{FV}(p_i)]$ for each i and $\Gamma \cup \{x : B_x \mid x \in \mathsf{FV}(p_i)\} \vdash P_i : A$ for all i. Then the result follows from the induction hypothesis and (MchR).

□

Theorem C.15 (Preservation on the Left). *Suppose* Γ *and* Δ *are disjoint, consistent variable environments. If* $M \rhd N$ *and* $\Gamma, M : A \vdash \Delta$ *then* $\Gamma, N : A \vdash \Delta$.

Proof. We prove the result for all $\mathcal{E}$, redexes P and contractions Q such that $M = \mathcal{E}[P] \rhd \mathcal{E}[Q] = N$, by induction on $\mathcal{E}$.

- When the context is just a hole, we proceed to analyse the redex.
  - **(Beta)** In this case, P has shape $(\lambda x.\, P')\, V$ and $Q = P'[V/x]$. It follows from inversion that either (1) there is a type B such that $\Gamma \vdash \lambda x.\, P' : B \rightarrowtail A$ and $\Gamma, V : B \vdash \Delta$, or (2) $\Gamma, \lambda x.\, P' : \mathsf{Ok} \rightarrowtail A \vdash \Delta$. In case (1), it follows by inversion that there are types B_1 and

B_2 such that $\Gamma, P' : B_2 \vdash x : B_1$ and $\Gamma \vdash B_1 \rightarrowtail B_2 \sqsubseteq B \rightarrowtail A$ (the other possibility is excluded by the consistency of the constraints). Then it follows from the substitution lemma that $\Gamma, P'[V/x] : B_2 \vdash \Delta$. By the closedness of the constraints, $\Gamma \vdash A \sqsubseteq B_2$ and so the desired conclusion follows by (SubL). By the Progress theorem, case (2) is impossible unless Δ is of shape $z : \mathsf{Ok}$ for some z. In this case, the conclusion follows immediately by (VarK).

(Delta), (Fix) By inversion, these cases are impossible unless Δ has shape $z : \mathsf{Ok}$ for some variable z and then the conclusion follows immediately by (VarK).

(Match) In this case, P has shape $\mathsf{match}\, c(V_1, \ldots, V_n)\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\}$ and one of the alternatives has shape $c(x_1, \ldots, x_n) \mapsto P$ and $Q = P[V_i/x_i \mid 1 \leq i \leq k]$. It follows from inversion that either Δ is of shape $z : \mathsf{Ok}$, in which case the conclusion is immediate by (VarK), or there are a family of types B_x, one for each pattern-bound variable x such that $\Gamma, P : A \vdash x : B_x$ (for each $x \in \{x_1, \ldots, x_n\}$) and, either $\Gamma, c(V_1, \ldots, V_n) : c(B_{x_1}, \ldots, B_{x_n}) \vdash \Delta$ or $\Gamma, P : A \vdash \Delta$. In the former case, it follows from inversion and the consistency of the constraints in Γ that there is j such that $\Gamma, V_j : B_{x_j} \vdash \Delta$. Then is follows from the substitution lemma that $\Gamma, P[V_j/x_j] \vdash \Delta$. Then it follows from the trivial substitution lemma, by induction on n, that $\Gamma, P[V_i/x_i \mid 1 \leq i \leq n] : A \vdash \Delta$ as required.

- When the context is of shape $\mathcal{E}\, M'$, it follows from inversion that either (0) Δ has shape $z : \mathsf{Ok}$ for some variable z, (1) there is a type B and $\Gamma \vdash \mathcal{E}[P] : B \rightarrowtail A$ and $\Gamma, M' : B \vdash \Delta$, or (2) $\Gamma, \mathcal{E}[P] : \mathsf{Ok} \rightarrowtail A \vdash \Delta$. In case (0) the result is immediate by (VarK). In case (1), we obtain $\Gamma \vdash \mathcal{E}[Q] : B \rightarrowtail A$ by preservation on the right and the result follows by (AppL). In case (2), we obtain the result from the induction hypothesis and (FunK).
- When the context has shape $(\lambda x.\, M')\, \mathcal{E}$, it follows from inversion that either (0) Δ has shape $z : \mathsf{Ok}$, or (1) there is a type B and $\Gamma \vdash \lambda x.\, M' : B \rightarrowtail A$ and $\Gamma, \mathcal{E}[P] : B \vdash \Delta$, or (2) $\Gamma \vdash \lambda x.\, M' : \mathsf{Ok} \rightarrowtail A \vdash \Delta$. In case (0) the result is immediate by (VarK). In case (2), the result follows by the induction hypothesis and (AppL). In case (3), the result follows from the induction hypothesis and (FunK).
- When the context has shape $\mathsf{match}\, \mathcal{E}\, \mathsf{with}\, \{|_{i=1}^{k} (p_i \mapsto P_i)\}$ it follows from inversion that either Δ has shape $z : \mathsf{Ok}$ or there is a family of types B_x (one for each pattern-bound variable x) such that $\Gamma, P_i : A \vdash x : B_x$ and, for each i, either $\Gamma, \mathcal{E}[P] : p_i[B_x/x \mid x \in \mathsf{FV}(p_i)] \vdash \Delta$ or $\Gamma, P_i : A \vdash \Delta$. Then the result follows from the induction hypothesis and (MchL).

□

D ADDITIONAL MATERIAL IN SUPPORT OF SECTION 6

In this appendix, we give the proofs of Theorem 6.4, showing that two-sided judgements of the specified form are subsumed by one-sided judgements, and Theorem 6.6, showing syntactic soundness of the one-sided system. The latter requires first establishing a kind of inversion lemma for values, that shows that they can only be given the types one would expect, and then substitution lemmas, and preservation.

D.1 Proof of Theorem 6.4

Proof. The proof is by induction on $\Gamma, M : A \vdash \Delta$.

(Id) In this case, there are two cases. If M is some variable x and $x : A \in \Delta$, then we may conclude $\Gamma \cup \Delta^c \vdash x : A^c$ by (Id). Otherwise, M is not a variable, but there is a variable typing $x : A$ in both Γ and Δ. In that case, $\Gamma \cup \Delta^c$ will contain a contradiction, and the result follows by (Contra).

(LetL1) In this case M is of shape $\mathsf{let}\,(x,y) = N\,\mathsf{in}\,P$ and we may assume (i) $\Gamma \cup \Delta^c \vdash P : A^c$. Then the result follows from (Let3) with A set to A^c

(LetR) In this case M is $\mathsf{let}\,(x_1,x_2) = N\,\mathsf{in}\,P$ and we can assume (i) $\Gamma \cup \Delta^c \vdash N : B \times C$ and (ii) $\Gamma \cup \Delta^c,\ x : B,\ y : C \vdash P : A$. Then the result follows immediately from (Let1).

(LetL2) In this case M is of shape $\mathsf{let}\,(x,y) = N\,\mathsf{in}\,P$ and we can assume (i) $\Gamma,\ \Delta^c\ x : B^c \vdash P : A^c$, (ii) $\Gamma \cup \Delta^c,\ y : C^c \vdash P : A^c$ and (iii) $\Gamma \cup \Delta^c \vdash M : (B \times C)^c$. The conclusion follows immediately by instantiating A in (Let2) by A^c.

(IfZR) In this case, M is of shape $\mathsf{if}\,N\,\mathsf{then}\,P_1\,\mathsf{else}\,P_2$ and we may assume (i) $\Gamma \cup \Delta^c \vdash N : \mathsf{Nat}$, (ii) $\Gamma \cup \Delta^c \vdash P_1 : A$, (iii) $\Gamma \cup \Delta^c,\ x : \mathsf{Nat} \vdash P_2 : A$. Then the desired conclusion follows immediately from (ii) and (iii) by (IfZ1).

(IfZL1) In this case, M is of shape $\mathsf{if}\,N\,\mathsf{then}\,P_1\,\mathsf{else}\,P_2$ and we may assume (i) $\Gamma \cup \Delta^c \vdash N : \mathsf{Nat}^c$. Then the result follows immediately by (IfZ2).

(IfZL2) In this case, M is of shape $\mathsf{if}\,N\,\mathsf{then}\,P_1\,\mathsf{else}\,P_2$ and we may assume (i) $\Gamma \cup \Delta^c \vdash P_1 : A^c$ and (ii) $\Gamma \cup \Delta^c,\ x : \mathsf{Nat} \vdash P_2 : A^c$. Then the result follows immediately by (IfZ1).

(CompL) In this case, we may assume $\Gamma \cup \Delta^c \vdash M : A$ and then the result is immediate.

(CompR) In this case, we may assume $\Gamma \cup \Delta^c \vdash M : A^c$ and again, the result is immediate.

(Disj) In this case, we may assume $\Gamma \cup \Delta^c \vdash M : B$ and $A \parallel B$. Then the result follows from (Dis).

(AppL) In this case, M is of shape $P\,N$ and we may assume (i) $\Gamma \cup \Delta^c \vdash P : B \rightarrowtail A$ and (ii) $\Gamma \cup \Delta^c \vdash N : B^c$. Recall that $B \rightarrowtail A$ is just an abbreviation for $B^c \to A^c$ in the one-sided system. Hence, the result follows immediately from (App).

(OkApL1) In this case, M is of shape $P\,N$ and we may assume (i) $\Gamma \cup \Delta^c \vdash P : (\mathsf{Ok} \rightarrowtail A)^c = (\mathsf{Ok}^c \to A^c)^c$. Therefore, the result follows from (App2).

(OkApL2) In this case, M is of shape $P\,N$ and we may assume (i) $\Gamma \cup \Delta^c \vdash N : \mathsf{Ok}^c$. Then the desired conclusion follows immediately by (App3).

(OkL) In this case we may assume $\Gamma \cup \Delta^c \vdash M : \mathsf{Ok}$. Hence, the result follows from (OkC2).

(OkR) In this case the result follows immediately from (Ok).

(OkSL),(SuccL) In these cases, M is of shape $\mathtt{succ}(N)$ and we may assume $\Gamma \cup \Delta^c \vdash N : \mathsf{Nat}^c$. Therefore, the results follow from (Succ2).

(OkPrL) In this case, M is of shape (N_1, N_2) and we may assume $\Gamma \cup \Delta^c \vdash M_i : \mathsf{Ok}^c$. Then the result follows immediately by (Pair2).

(PairL) In this case, M is of shape (N_1, N_2), A is of shape $A_1 \times A_2$. Let us consider the $i = 1$ case, the other is symmetrical. We may assume that $\Gamma \cup \Delta^c \vdash N_1 : A_1^c$. Then the result follows from (Pair3).

(AbsR) The result follows from the induction hypothesis and (Abs).

(AbnR) In this case, M is of shape $\lambda x.\,P$ and A is of shape $B \rightarrowtail A$. We may assume $\Gamma \cup \Delta^c,\ x : B^c \vdash M : A^c$. Since, in the one-sided system, $B \rightarrowtail A$ is an abbreviation for $B^c \to A^c$, the result follows from (Abs).

(AppR) The result follows from the induction hypotheses and (App1).

(FixR) The result follows from the induction hypotheses and (Fix).

(SuccR) The result follows from the induction hypotheses and (Succ1).

(ZeroR) The result follows from the induction hypotheses and (Zero).

□

D.2 Proof of Soundness

Lemma D.1 (Value Inversion). *Suppose* $\vdash V : A$.

- *If* V *is of shape* $\mathtt{zero}$ *then* $A = \mathsf{Nat}$ *or* A *is of shape* B^c *with* $B \parallel \mathsf{Nat}$
- *If* V *is of shape* $\mathtt{succ}(W)$ *then* $A = \mathsf{Nat}$ *or* A *is of shape* B^c *with* $B \parallel \mathsf{Nat}$.

- *If V is of shape (W_1, W_2) then either:*
  - *A is of shape $B \times C$*
  - *or, A is of shape B^c and B is not a pair type nor* Ok.
  - *or, A is of shape $(B_1 \times B_2)^c$ and there is i with $\vdash W_i : B_i^c$*
- *If V is of shape $\lambda x.M$ then either:*
  - *there are types B_1 and B_2 such that $A = B_1 \to B_2$ and $x : B_1 \vdash M : B_2$*
  - *or, A is of shape B^c with B not an arrow type nor* Ok.

Proof. The proof is by induction on the derivation. In cases (OkC), (Contra) and (Var), the conclusion is immediate because the rules only apply to judgements with a non-empty environment. In cases (Fix),(Let1), (Let2), (App1), (App2), (App3), (IfZ1) and (IfZ2) the conclusion is immediate since the subject is not a value.

(OkC2) We suppose the induction hypothesis, but then we obtain a contradiction because, by case analysis on V, it follows that Ok^c cannot be Ok^c. Hence, the result follows vacuously.

(Dis) In this case, A has shape A'^c and we assume $B \mid\mid A'$. We proceed by case analysis on V. Note that, B cannot be of shape B'^c.

- If V is a numeral $\underline{n}$, then it follows from the induction hypothesis that B must be Nat. Therefore, A is of shape A'^c with $A' \mid\mid \mathsf{Nat}$, as required.
- If V is a pair (W_1, W_2), then it follows from the induction hypothesis that B must have shape $B_1 \times B_2$. Therefore, A' is not a pair type nor Ok.
- If V is an abstraction $\lambda x.P$, then it follows from the induction hypothesis that B must have shape $B_1 \to B_2$. Therefore A' is not an arrow type nor Ok.

(Zero),(Succ1) In these cases, V is a numeral and A is Nat.

(Succ2) In this case, V is a numeral. We assume the induction hypothesis, but then we obtain a contradiction because Nat^c is not of shape B^c with $B \mid\mid \mathsf{Nat}$.

(Abs) In this case, V is an abstraction and A has shape $B_1 \to B_2$.

(Pair1) In this case, V is of shape (W_1, W_2) and A is of shape $A \times B$ and $\Gamma \vdash W_1 : A$ and $\Gamma \vdash W_2 : B$

(Pair2) In this case, we suppose the induction hypothesis, but then we obtain a contradiction because the type of W_i cannot be Ok^c.

(Pair3) In this case, V is of shape (W_1, W_2) and A is of shape $(B_1 \times B_2)$ and there is an i such that $\Gamma \vdash W_i : B_i^c$.

□

Lemma D.2. *Suppose Γ is a type environment and V a closed value and $\Gamma \vdash M : A$.*

- *If $x : B \in \Gamma$ and $\Gamma \setminus \{x : B\} \vdash V : B$ then $\Gamma \setminus \{x : B\} \vdash M[V/x] : A$.*
- *If $x \notin \mathsf{Subjects}(\Gamma)$, then $\Gamma \vdash M[V/x] : A$.*

Proof. The proof is by induction on the derivation.

(Ok) In this case, $\Gamma \vdash M[V/x] : \mathsf{Ok}$ by (Ok).

(OkC1) In this case, Γ contains $y : \mathsf{Ok}^c$ and we consider two cases. If $x = y$, then we may assume $\Gamma \setminus \{x : B\} \vdash V : \mathsf{Ok}^c$ and this gives the result since $x[V/x] = V$. Otherwise, the result follows from (OkC1).

(Zero) In this case, the result follows again from (Zero).

(Contra) This rule does not apply when Γ is a type environment.

(Var) In this case, Γ contains $y : A'$. If $x = y$ then $A = A'$ and we may assume $\Gamma - \{x : B\} \vdash V : A$. Then this gives the result since $x[V/x] = V$. Otherwise, the result follows from (Var) since $y[V/x] = y$.

In the remaining cases, the result follows from the induction hypotheses and the fact that bound variables in the conclusion may be assumed distinct from x. □

Theorem D.3. *If* $\vdash M : A$ *and* $M \rhd N$ *then* $\vdash N : A$.

Proof. The proof is by induction on the derivation. The cases (OkC1), (Contra) and (Var) do not apply since there are no assumptions.

(Ok) In this case, the result follows immediately from (Ok).

(OkC2) It follows from the induction hypothesis that $\vdash N : \mathsf{Ok}^c$ and the result follows from (OkC2).

(Dis) In this case, A is of shape A'^c. Suppose $A \parallel B$. It follows from the induction hypothesis that $\vdash N : B$ and hence the result follows from (Dis).

(Zero) In this case, M is $\mathtt{zero}$ and we obtain a contradiction from $M \rhd N$.

(Succ1) In this case, M is of shape $\mathtt{succ}(P)$ and $A = \mathsf{Nat}$. Thus it must be that P makes a step to some Q. It follows from the induction hypothesis that $\vdash Q : \mathsf{Nat}$ and the result follows from (Succ1).

(Succ2) In this case, M is of shape $\mathtt{succ}(P)$ and thus it must be that P makes a step to some Q. It follows from the induction hypothesis that $\vdash Q : \mathsf{Nat}^c$ and thus the result follows from (Succ2).

(Abs) In this case, M does not make a step.

(Fix) In this case, M is of shape $\mathsf{fix}\, y.\, P$ and thus $N = P[\mathsf{fix}\, y.\, P/y]$. In this case we may assume that $\Gamma, x : A \vdash M : A$ (this corresponds to proving a strengthened induction hypothesis) and so it follows from the substitution lemma that $\Gamma \vdash P[\mathsf{fix}\, y.\, P/y] : A$, as required.

(Let3) In this case, M has shape $\mathsf{let}\,(x, y) = Q\,\mathsf{in}\,P$ and there are two cases. If Q makes a step to some Q' then the result follows from the induction hypothesis. Otherwise, it must be that Q is a pair (V, W) and $N = P[V/x, W/y]$. Since x and y do not occur in Γ, the result follows from the substitution lemma.

(Let2) In this case, M has shape $\mathsf{let}\,(x, y) = Q\,\mathsf{in}\,P$ and there are two cases. If Q makes a step to some Q' then the result follows from the induction hypothesis. Otherwise, it must be that Q is a pair (V, W) and $N = P[V/x, W/y]$. Hence, it follows by value inversion on the first premise that either $\Gamma \vdash V : B_1^c$ or $\Gamma \vdash W : B_2^c$. In either case, we can use the corresponding second premise and the substitution lemma (both parts) to conclude.

(Let1) In this case, M has shape $\mathsf{let}\,(x, y) = Q\,\mathsf{in}\,P$ and there are two cases. If Q makes a step to some Q' then the result follows from the induction hypothesis. Otherwise, it must be that Q is a pair (V, W) and $N = P[V/x, W/y]$. It follows from value inversion that $\Gamma \vdash V : B$ and $\Gamma \vdash W : C$ and so we can conclude using the substitution lemma.

(App1) In this case, M has shape PQ. There are two cases. If P or Q makes a step, then the result follows from the induction hypothesis. Otherwise, it must be that P is an abstraction $\lambda x.\, P'$ and Q is a value. By value inversion, it must be that $x : B \vdash P' : A$ and then the result follows from the substitution lemma.

(App2) In this case, M has shape PQ. There are three cases. If P makes a step, then the result follows from the induction hypothesis. If Q makes a step, the result follows immediately from (App2). Otherwise P must be an abstraction $\lambda x.\, P'$ but it follows from value inversion that this is impossible with type $(\mathsf{Ok}^c \to A)^c$.

(App3) In this case, M has shape PQ. There are three cases. If Q makes a step, then the result follows from the induction hypothesis. If P makes a step, the result follows immediately from (App3). Otherwise, it must be that Q is a value, but it follows from value inversion that this is impossible with type Ok^c.

(Pair1) In this case, M is a pair (P, Q) and A has shape $B_1 \times B_2$. It can only be that either P makes a step or Q makes a step, and then the result follows from the induction hypothesis in each case.

(Pair2) In this case, M is a pair (P_1, P_2) and $i \in \{1, 2\}$. If P_i makes a step, then the result follows from the induction hypothesis; otherwise the result follows immediately from (Pair2).

(Pair3) In this case, M is a pair (M_1, M_2) and $i \in \{1, 2\}$ and A has shape $(A_1 \times A_2)^c$. If M_i makes a step, then the result follows from the induction hypothesis; otherwise the result follows immediately from (Pair3).

(IfZ1) In this case, M is of shape $\mathsf{if}\, Q \,\mathsf{then}\, P_1 \,\mathsf{else}\, P_2$. There are two cases. If Q makes a step, then the result follows from the induction hypothesis. Otherwise, M is a numeral. However, it follows from value inversion that this is impossible with type Nat^c.

(IfZ2) In this case, M is of shape $\mathsf{if}\, Q \,\mathsf{then}\, P_1 \,\mathsf{else}\, P_2$. There are two cases. If Q makes a step, then the result follows from the induction hypothesis. Otherwise, M is a numeral. If $M = \underline{0}$, then $M \rhd P_1$ and the result follows from the first premise. Otherwise, $M \rhd P_2$ and the result follows from the second premise.

□

D.2.1 Proof of Theorem 6.6 (One-Sided Syntactic Soundness).

Proof. Suppose $\vdash M : \mathsf{Ok}^c$. For the purpose of obtaining a contradiction, suppose M reaches a value V. Then it follows from preservation that $\vdash V : \mathsf{Ok}^c$, but this contradicts the value inversion lemma. □