Programming Research Group

SEMANTICS AND TYPE CHECKING OF DEPENDENTLY-TYPED LAZY FUNCTIONAL PROGRAMS

Yorck Hünke

PRG-RR-04-15

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

We present a denotational semantics, type checking algorithm and soundness result for a language with dependent function and record types, dependent case expressions, a fixpoint combinator and a type of all types. The design of our algorithm is guided by key ideas from related work on the semantics of dependent types in domain theory, normalisation of typed lambda terms with sums, and subtyping of recursive types. We address a subtlety that arises with respect to the soundness of checking case expressions if the underlying language is based on a non-strict evaluation order. Subtyping of recursive types is extended to take into account functions occurring in types. The expressiveness of our language comes at the cost of undecidable type checking; we discuss the limitations of our algorithm, which arise mainly from the interaction of case expressions and fixpoints.

1 Introduction

One of the main benefits of the functional programming language Haskell [26] is its strong type system, which helps programmers detect many errors at compile-time. But the type system also prevents programmers from writing certain "sensible" programs. To overcome this limitation, a wide range of extensions to Haskell's type system have been proposed and implemented, such as type classes with functional dependencies [19] and generic programming [18].

Yet limitations remain, and recently suggestions have been made to extend Haskell with *dependent types*, which would make the language much more flexible. Although dependent types have been widely investigated in the context of logical frameworks [6, 11, 27], they are still perceived as somewhat inaccessible by many functional programmers, and a gap remains to be bridged between the two communities.

This paper contributes towards bridging this gap by synthesising key ideas from previous work, resulting in a denotational semantics, type checking algorithm and soundness result for a practical dependently-typed functional programming language that is type-safe under lazy evaluation. We now introduce the work that we synthesise, and outline how we adapt it to our setting.

1.1 A language with dependent types

Augustsson [4] took an important step towards making dependently-typed functional programming accessible, by combining dependently-typed constructs from type theory with general recursion and presenting the resulting language Cayenne in a notation familiar to functional programmers.

Dependent function types allow the result type of functions to depend on the value of their argument. The branches of case expressions can have different types depending on the scrutinised value. This allows a natural and type-safe definition of the function *printf*, for example, whose first argument is a formatting string that determines the number and types of further arguments.

$$printf :: (format : String) \rightarrow printfType\ format$$

The type-level function *printfType* :: *String* $\rightarrow \star$ takes the formatting string that is passed to *printf* and returns the corresponding result type of *printf*. This particular example has also been implemented without dependent types [17], but it requires a lot of machinery to avoid the limitations of current type systems, whereas dependent types provide a natural solution.

Dependent records allow the types of later fields to depend on the value of earlier fields. This allows dynamic typing, for example, by letting the value of the first field determine the type of the second field. Cayenne's dependent records are also sufficiently powerful to serve as first-class modules.

1.2 Dependent types in domain theory

Although an elegant denotational semantics in domain theory [28] for a dependent type *system* had been known for some time [9], Coquand and Takeyama [13] recently gave the first type checking *algorithm* for a dependently-typed λ-calculus that has been proved correct with respect to such a semantics. Despite the logical inconsistency due to the presence of a type of all types *including itself* [8], their language is meaningful for programming because their soundness result implies that programs successfully type checked using their algorithm "cannot go wrong" in the style of Milner [23].

Coquand and Takeyama observe that the β-reduction rule

$$(\lambda x : a.e_1)\ e_2 \rightarrow e_1\ [e_2/x]$$

needed in their type checking algorithm is problematic with respect to their proposed semantics because the reduction forgets the type of the abstracted identifier. Intuitively, this rule should only be used for well-typed terms, where e_2 has type a, but this intuitive justification may look somewhat circular because typing itself uses computation rules, so a rigorous justification of this rule is desirable and given in [13].

1.3 Normalisation of typed λ-terms with sums

Normalisation by evaluation (NBE) and type-directed partial evaluation (TDPE) for simply-typed λ-calculus with sums have been investigated by Altenkirch, Dybjer, Hofmann and Scott [1] and Balat, Di Cosmo and Fiore [7], respectively. NBE relies on inverting an interpretation function (which identifies any two convertible syntactic terms with the same semantic value) such that semantic values are mapped to a representative syntactic term in normal form. TDPE is a program specialisation technique, and it is a specific form of NBE in the sense that it constructs code of compiled programs. The key idea of both techniques can essentially be thought of as an η-expansion guided by the structure of types.

1.4 Subtyping recursive types

Checking equality or subtyping of recursive types is non-trivial because a simple structural equality test does not account for syntactically different fixpoint expressions whose infinite expansions are identical. For example, the types $\mu t.\mathsf{Int} \to t$ and $\mu s.\mathsf{Int} \to \mathsf{Int} \to s$ both represent functions that take an infinite number of integer arguments.

Amadio and Cardelli [3] give an algorithm for subtyping recursive types over the following type language.

$$\alpha := t \mid \bot \mid \top \mid \alpha \to \beta \mid \mu t.\alpha$$

They prove soundness and completeness of their algorithm with respect to a model of infinite trees, which formalises the intuition that recursive types with identical infinite expansions should be equal.

1.5 Contributions of this paper

We synthesise and extend the previous work as follows.

- The language design and type system for Cayenne [4] lacks a semantic foundation and documented type checking algorithm. We give a denotational semantics (Sections 2–4) and a full type checking algorithm (Sections 5–7) for a language similar to core Cayenne, and present a soundness result for the algorithm with respect to our semantics. All proofs can be found in Appendix B.

- Coquand and Takeyama [13] provide no constructs for dependent records, dependent case expressions or recursion, which are essential for practical programming. The main focus of this paper is on the more interesting extensions for case expressions and recursion. The treatment of dependent records is relatively straightforward but rather heavy on notation; it is not necessary for an understanding of the main focus of this paper and can safely be skipped upon first reading.

- The concept of η-expansion based on the structure of types as used in NBE and TDPE for typed λ-calculus with sums [1, 7] is problematic in the presence of recursive types. Inspired by the structure of normal forms and a notion of constraints defined in the NBE and TDPE techniques, we show how to deal with commuting conversions of case expressions [15] in our setting using a restricted form of η-expansion guided by the structure of *terms*.

- Our semantics exposes a subtlety with respect to checking case expressions under a non-strict evaluation order. Consider the equality for distributing a function through a case expression.

  $$f(\mathit{case}\ e\ \mathit{of} \ldots b_i \ldots) \ = \ \mathit{case}\ e\ \mathit{of} \ldots (f\ b_i) \ldots$$

  This rule may need to be applied during type checking, but it does not hold in general for non-strict f. We show how to integrate a simple strictness analysis into our algorithm in order to ensure the soundness of such rules under lazy evaluation.

- Amadio and Cardelli's algorithm for subtyping recursive types [3] covers only a restricted type language. We show how their ideas can be extended to take into account function definitions and applications within types. For example, our algorithm can identify the two list types

$$\begin{aligned} list_1 &= \mu\ l : \star \to \star\ .\ \lambda a : \star\ .\ \{\mathsf{Nil},\ \mathsf{Cons}\ a\ (l\ a)\} \\ list_2 &= \lambda a : \star\ .\ \mu\ l : \star\ .\ \{\mathsf{Nil},\ \mathsf{Cons}\ a\ l\} \end{aligned}$$

where the curly brackets denote a (non-recursive) data type definition, and $\star$ is the type of all types.

2 Semantic model

A suitable and convenient choice of semantic model for our language is to use the well-established machinery of domain theory, originally developed by Scott [28]. This has become a standard model for languages with general recursion, because it gives solutions to domain equations where set theory fails and it can model partial and infinite elements. No deep understanding of domain theory is required to follow this paper, but we will assume a basic familiarity with the subject; a good introduction to denotational semantics and domain theory can be found in [24] and [16], respectively.

This section introduces the domain equations, the interpretation of types and the typing relation given in [13] for a dependently-typed λ-calculus, and extends them with data types and dependent records.

2.1 Domain equations

Our semantic model is given by a solution to the following domain equation, where $\oplus$ denotes smashed sum.

$$\mathsf{D} \quad = \quad \mathsf{T} \oplus [\mathsf{D} \to \mathsf{D}] \oplus (\textstyle\bigoplus_{c \in Con} \mathsf{Inj}_c\ \mathsf{D}) \oplus \mathsf{D}^{Ide}$$

Expressions in our language are assigned denotations in the domain D. Type expressions are mapped into the subdomain T, defined further below. The subdomain $[\mathsf{D} \to \mathsf{D}]$ contains the continuous functions from D to itself, and $\mathsf{Inj}_c\ \mathsf{D}$ contains injections of elements $d \in \mathsf{D}$ into data types, where c are constructor labels taken from a (countable) set Con. The subdomain D^{Ide} contains records represented by functions from a (countable) set Ide of identifiers to D.

The notation $\mathsf{Inj}_c\ \mathsf{D}$ stands for the lifting of the domain D, and is used instead of the conventional notation $\mathsf{D}_\uparrow$ to be able to distinguish the different liftings for each of the constructors. A typical element of $\mathsf{Inj}_c\ \mathsf{D}$ is written $\mathsf{Inj}_c\ d$, where $d \in \mathsf{D}$. Elements of D^{Ide} are written as ϱ, and update of a field identifier x with a value $d \in \mathsf{D}$ in a tuple ϱ is written $(\varrho, x = d)$.

The domain T of (codes for) types is given by a solution to the following domain equation.

$$\mathsf{T} \quad = \quad \mathsf{V} \oplus \mathsf{Prod}\ \mathsf{T}\ [\mathsf{D} \to \mathsf{T}] \oplus \mathsf{Data}\ (\mathsf{Def}\ \mathsf{T})^{Con} \oplus (\textstyle\bigoplus_{xs \in (Ide^{\pm})^*} \mathsf{Sig}_{xs}\ (sig\ xs))$$

$$\begin{array}{llcl} & sig\ \langle x^-, xs\rangle & = & \mathsf{T} \times [\mathsf{D} \to (sig\ xs)] \\ \text{where} & sig\ \langle x^+, xs\rangle & = & \mathsf{T} \times \mathsf{D} \times (sig\ xs) \\ & sig\ \langle\rangle & = & \bot \end{array}$$

The subdomain V has only one non-bottom element V representing the type of all types including itself, $\mathsf{Prod}\ \mathsf{T}\ [\mathsf{D} \to \mathsf{T}]$ contains dependent function types, and $\mathsf{Data}\ (\mathsf{Def}\ \mathsf{T})^{Con}$ contains data types. The subdomain $\mathsf{Sig}_{xs}\ (sig(xs))$ contains record types.

The notation $\mathsf{Prod}\ \mathsf{T}\ [\mathsf{D} \to \mathsf{T}]$ stands for the lifting of the product of domains T and $[\mathsf{D} \to \mathsf{T}]$, conventionally written as $(\mathsf{T} \times [\mathsf{D} \to \mathsf{T}])_\uparrow$. A typical element of this domain is written as $\mathsf{Prod}\ u\ f$, with $u \in \mathsf{T}$ and $f \in [\mathsf{D} \to \mathsf{T}]$. Elements of $\mathsf{Data}\ (\mathsf{Def}\ \mathsf{T})^{Con}$ are written as $\mathsf{Data}\ \Delta$, where $\Delta \in (\mathsf{Def}\ \mathsf{T})^{Con}$.

The domains for record types are indexed by a finite sequence xs of distinct identifiers, representing field names of records. The superscripts $+$ and $-$ of identifiers in xs indicate respectively whether a field value is present in the signature (manifest field) or not (variable field). For convenience, a typical element of $\mathsf{Sig}_{xs}\ (sig(xs))$ is written as $[xs : s]$, where $s \in sig(xs)$. A typical element of $\mathsf{Sig}_{\langle x^-, xs\rangle}\ (sig(x^-, xs))$ is written as $[x : u, xs : f]$, where $u \in \mathsf{T}$ and $f \in [\mathsf{D} \to (sig\ xs)]$, a typical element of $\mathsf{Sig}_{\langle x^+, xs\rangle}\ (sig(x^+, xs))$ is written $[x : u = d, xs : s]$, where $u \in \mathsf{T}$, $d \in \mathsf{D}$ and $s \in [sig\ xs]$, and the only element of $\mathsf{Sig}_{\langle\rangle}\ (sig\langle\rangle)$ is written as $[]$.

2.2 Interpretation of type codes

Elements in the domain T are only codes for types, so what are types themselves and how do they relate to the codes? Types are often thought of as sets of elements. We extend this notion by defining types to be subdomains of the domain D, so that elements of D contained in a particular subdomain have that subdomain as its type. For example, the element V represents the type of all types, and is thus interpreted as the subdomain T of D.

We now formalise this idea. For convenience, we work with the domain $\mathsf{Fp}(\mathsf{D})$ of *finitary projections* over D, which is isomorphic to the domain of subdomains of D (see [16]).

$$\mathsf{Fp}(\mathsf{D}) = \{p : \mathsf{D} \to \mathsf{D} \mid p \sqsubseteq id,\ p \circ p = p,\ \text{range}(p) \text{ is a domain}\}$$

Our interpretation of type codes is defined as a continuous map $u \mapsto \mathsf{p}_u$ from type codes to actual types, defined as a solution $\mathsf{p}_{(-)} \in [\mathsf{T} \to \mathsf{Fp}(\mathsf{D})]$ of the following recursive equations, where any case that is not explicitly specified is defined to be $\bot$.

$$\begin{array}{lcll} \mathsf{p}_{\mathsf{V}}\ u & = & u & (u \in \mathsf{T}) \\ \mathsf{p}_{\mathsf{Prod}\ u\ f}\ g & = & \lambda d.\mathsf{p}_{(f\ d')}(g\ d') & (g \in [\mathsf{D} \to \mathsf{D}]) \\ & & \quad \text{where } d' = \mathsf{p}_u d & \\ \mathsf{p}_{\mathsf{Data}\ \Delta}(\mathsf{Inj}_c\ d) & = & \mathsf{Inj}_c\ (\mathsf{p}_u d) & (c\Delta = \mathsf{Def}\ u) \\ \mathsf{p}_{[x:u, xs:f]}\varrho & = & (\mathsf{p}_{[xs:f(\mathsf{p}_u x\varrho)]}\varrho, x = \mathsf{p}_u x\varrho) & (\varrho \in \mathsf{D}^{Ide}) \\ \mathsf{p}_{[x:u=d, xs:s]}\varrho & = & (\mathsf{p}_{[xs:s]}\varrho, x = \mathsf{p}_u(d \sqcap x\varrho)) & (\varrho \in \mathsf{D}^{Ide}) \end{array}$$

Intuitively, each finitary projection p_u is a function that *restricts* elements in D to the type represented by the code $u \in \mathsf{T}$. The range of a finitary projection p_u corresponds to the subdomain represented by u. For example, the range of $\mathsf{p_V}$ is the subdomain T of D. For function types, the result of restricting a function $g \in [\mathsf{D} \to \mathsf{D}]$ using a code $\mathsf{Prod}\ u\ f$ is a function that takes an input d, restricts it to the input type u, applies the function g to the restricted input, and restricts that result to the output type code, namely the result of applying f to the restricted input $\mathsf{p}_u d$. To restrict an injection $\mathsf{Inj}_c\ d$ using $\mathsf{Data}\ \Delta$, we restrict the injected element d using the type code u obtained from Δ, provided the constructor label c is contained in the data type ($c\Delta = \mathsf{Def}\ u$), and otherwise return $\perp$.

Records are restricted to a type code $[x : u, xs : f]$ by restricting the field x in ϱ to type code u, and restricting the remaining fields with the type code obtained by applying f to the restriction of field x. Records are restricted to a type code $[x : u = d, xs : s]$ by taking the greatest lower bound of the field x in ϱ and of d (to ensure well-definedness of $\mathsf{p}_{(-)}$), restricting the result to type code u, and restricting the remaining fields with the type code $[xs : s]$.

2.3 Typing relation

We define our typing relation $d : u$ to mean that $d \in \mathrm{range}(\mathsf{p}_u)$, which is equivalent to $\mathsf{p}_u d = d$. This means that an element $d \in \mathsf{D}$ has the type represented by code $u \in \mathsf{T}$ provided p_u leaves d unchanged.

3 Expressions

This section describes the syntax of our language, its semantics in our domain D, and a semantic type checking judgement that will be used in later sections to prove soundness of our syntactic type checking algorithm.

3.1 Syntax

The syntax of expressions is described by the following grammar e. For readability, a stands for expressions that are intended to be types.

$$
\begin{aligned}
e &:= x \mid \star \mid \lambda x : a.e \mid \forall x : a.a \mid e\ e \mid \mu x : a.e \\
&\mid\ c\ e\ a \mid \{d\} \mid \mathit{case}\ a\ b\ e \\
&\mid\ \langle r :: a \rangle \mid \langle s \rangle \mid e.x \\
d &:= \bullet \mid c\ a, d \\
b &:= \bullet \mid c\ x.e, b \\
r &:= \bullet \mid x = e, r \\
s &:= \bullet \mid x : a, s \mid x = e, s
\end{aligned}
$$

Our grammar contains identifiers x, a type of all types including itself $\star$, typed abstractions $\lambda x : a.e$, dependent function types $\forall x : a.a$, applications $e\ e$, a typed fixpoint operator $\mu x : a.e$, constructors $c\ e\ a$, data types $\{d\}$ and case expressions $\mathit{case}\ a\ b\ e$.

Constructors $c\ e\ a$ inject an expression e using a constructor name $c \in Con$, and are annotated with their result type a to allow the same name c to be used in distinct data type definitions. Case expressions $case\ a\ b\ e$ analyse the expression e (which we call the *scrutinee*), have branches b and are annotated with their result type a. Bullets $\bullet$ denote empty sequences. We require that the sequence of constructors in a single data type or case branch definition are distinct.

The grammar e also contains dependent records $\langle r :: a \rangle$, signatures $\langle s \rangle$ and record field selection $e.x$. We require that all identifiers in $dom(r)$ and $dom(s)$ are distinct.

There are two main differences between our language and core Cayenne. First, we have a type of all types instead of a type hierarchy. A type of all types destroys logical consistency [8], but so does general recursion, which we do not wish to dispense with. Our semantics shows that this design decision does not affect type-safety. Second, our data types are non-recursive, and record fields are not bound within other fields of the same record. Recursive data types and mutually recursive record fields can be obtained by placing them inside a fixpoint expression.

The syntactic sugar available in Cayenne carries over to our language.

3.2 Semantics

The domain of semantic environments is $\mathsf{Env} = (\mathsf{D} \times \mathsf{T})^{Ide}$. For $\nu \in \mathsf{Env}$, we write $\nu(x) = d : u$ to mean $\nu(x) = (d, u)$. Given $u \in \mathsf{T}$ and $d \in \mathsf{D}$, environment update is defined by

$$(\nu, x : u = d)(x) = \mathsf{p}_u d : u$$
$$(\nu, x : u = d)(y) = \nu(y) \qquad (y \neq x).$$

The semantics of expressions is given by two continuous functions $\nu \mapsto e\nu : \mathsf{Env} \to \mathsf{D}$ and $\nu \mapsto (e \mid \nu) : \mathsf{Env} \to \mathsf{T}$. The former is the denotation of e under ν, the latter denotes the type of e under ν. Both are defined by induction on e, and given in Figure 1. We use auxiliary semantic functions defined in Figure 2, where cases not explicitly specified are equal to $\perp$, as before.

Lemma 1. *For all expressions e and $\nu \in \mathsf{Env}$, we have $e\nu : e \mid \nu$.*

3.3 Semantic type checking

Figure 3 defines a semantic version of type checking for expressions, which is the first step towards a syntactic version. The judgement $\nu \vdash e$ says that expression e is well-typed under semantic environment ν. These judgements will serve as the induction hypotheses in the soundness proof of our syntactic type checking judgement, thus avoiding the apparent circularity between the evaluation and typing rules described in Section 1.

4 Values

This section presents an intermediate step towards a syntactic version of type checking. It generalises expressions to values, introducing syntactic equivalents of the evident semantic

$$
\begin{aligned}
x\ \nu &= \mathsf{p}_u d \\
\star\ \nu &= \mathsf{V} \\
\lambda x : a.e\ \nu &= \lambda d.e\nu_d \\
\forall x : a.a_2\ \nu &= \mathsf{Prod}\ (\mathsf{p}_{\mathsf{V}} a\nu)\ (\lambda d.\mathsf{p}_{\mathsf{V}} a_2 \nu_d) \\
e_1\ e_2\ \nu &= (e_1\nu) \cdot (e_2\nu) \\
\mu x : a.e\ \nu &= \mathsf{fix}\,(\lambda d.\mathsf{p}_{\mathsf{p}_{\mathsf{V}} a\nu}(e\nu_d)) \\
(c\ e\ a)\ \nu &= \mathsf{p}_{\mathsf{p}_{\mathsf{V}} a\nu}(\mathsf{Inj}_c\ e\nu) \\
\{d\}\ \nu &= \mathsf{Data}\ (data\ d\ \nu) \\
(case\ a\ b\ e)\ \nu &= \mathsf{p}_{\mathsf{p}_{\mathsf{V}} a\nu}([\![b]\!]\ (e\nu)\ (e \mid \nu)\ \nu) \\
\langle r :: a \rangle \nu &= \tau\ (\mathsf{p}_{\mathsf{V}} a\nu)\ (r\nu) \\
e.x\ \nu &= \sigma_x\ (e\nu)\ (e \mid \nu) \\
\langle \bullet \rangle \nu &= [] \\
\langle x : a, s \rangle \nu &= [x : \mathsf{p}_{\mathsf{V}} a\nu, \\
&\qquad \lambda d.\langle s \rangle (\nu, x : \mathsf{p}_{\mathsf{V}} a\nu = d)] \\
\langle x = e, s \rangle \nu &= [x : e \mid \nu = e\nu, \\
&\qquad \langle s \rangle (\nu, x : e \mid \nu = e\nu)]
\end{aligned}
$$

$$
\begin{aligned}
x \mid \nu &= u \qquad \text{where } \nu(x) = d : u \\
\star \mid \nu &= \mathsf{V} \\
\lambda x : a.e \mid \nu &= \mathsf{Prod}\ (\mathsf{p}_{\mathsf{V}} a\nu)\ (\lambda d.e \mid \nu_d) \\
\forall x : a.a_2 \mid \nu &= \mathsf{V} \\
e_1\ e_2 \mid \nu &= (e_1 \mid \nu) * (e_2\nu) \\
\mu x : a.e \mid \nu &= \mathsf{p}_{\mathsf{V}} a\nu \\
(c\ e\ a) \mid \nu &= \mathsf{p}_{\mathsf{V}} a\nu \\
\{d\} \mid \nu &= \mathsf{V} \\
case\ a\ b\ e \mid \nu &= \mathsf{p}_{\mathsf{V}} a\nu \\
\langle r :: a \rangle \mid \nu &= \mathsf{p}_{\mathsf{V}} a\nu \\
e.x \mid \nu &= \Sigma_x\ (e\nu)\ (e \mid \nu) \\
\langle s \rangle \mid \nu &= \mathsf{V}
\end{aligned}
$$

$$
\text{where} \quad \nu_d = (\nu, x : \mathsf{p}_{\mathsf{V}} a\nu = d) \qquad \text{and} \qquad
\begin{aligned}
()\ \nu &= \bot \\
(x = e, r)\nu &= (r\nu, x = e\nu)
\end{aligned}
$$

Figure 1: Semantics of expressions

$$(-) \cdot (-) \ : \ \mathsf{D} \times \mathsf{D} \to \mathsf{D}$$
$$d_1 \cdot d_2 = d_1\, d_2, \quad \text{if } d_1 \in [\mathsf{D} \to \mathsf{D}]$$

$$(-) * (-) \ : \ \mathsf{D} \times \mathsf{D} \to \mathsf{T}$$
$$\mathsf{Prod}\ u\ f * d = f\ (\mathsf{p}_u\, d)$$

$$\mathsf{fix}\,(-) : [\mathsf{D} \to \mathsf{D}] \to \mathsf{D}$$
$$\mathsf{fix}\, g = \bigsqcup \{ g^n \bot \mid n \in \mathbb{N} \}$$

$$\begin{aligned} &data\ d\ (-) : \mathsf{Env} \to (\mathsf{Def}\ \mathsf{T})^{Con} \\ &data\ \bullet\ \nu = \bot \\ &data\ (c\ a, d)\ \nu = (data\ d\ \nu, c = \mathsf{Def}\ (\mathsf{p}_{\mathsf{V}}\, a\nu)) \end{aligned}$$

$$\begin{aligned} &[\![b]\!]\ (-)\ (-)\ (-) : \mathsf{D} \times \mathsf{T} \times \mathsf{Env} \to \mathsf{D} \\ &[\![\bullet]\!]\ d\ u\ \nu = \bot \\ &[\![c'\ x.e, b]\!]\ (\mathsf{Inj}_c\ d)\ (\mathsf{Data}\ \Delta)\ \nu = \\ &\qquad \begin{cases} e(\nu, x : u = d) & (c = c', c\Delta = \mathsf{Def}\ u) \\ [\![b]\!]\ (\mathsf{Inj}_c\ d)\ (\mathsf{Data}\ \Delta)\ \nu & (c \neq c', c\Delta = \mathsf{Def}\ u) \\ \bot & (c\Delta \neq \mathsf{Def}\ u) \end{cases} \end{aligned}$$

$$\begin{aligned} \tau\ (-)\ (-) &: \mathsf{T} \times \mathsf{D} \to \mathsf{D} \\ \tau\ [x : u, xs : f]\ \varrho &= (\tau\ [xs : f(\mathsf{p}_u x\varrho)]\ \varrho, x = \mathsf{p}_u x\varrho) \\ \tau\ [x : u = d, xs : s]\ \varrho &= (\tau\ [xs : s]\ \varrho, x = \mathsf{p}_u\, d) \end{aligned}$$

$$\begin{aligned} \sigma_x\ (-)\ (-) &: \mathsf{D} \times \mathsf{D} \to \mathsf{D} \\ \sigma_x\ \varrho\, [x : u, xs : f] &= \mathsf{p}_u x\varrho \\ \sigma_x\ \varrho\, [y : u, xs : f] &= \sigma_x\ \varrho\, [xs : f(\mathsf{p}_u y\varrho)] \quad (x \neq y) \\ \sigma_x\ \varrho\, [x : u = d, xs : s] &= \mathsf{p}_u(d \sqcap x\varrho) \\ \sigma_x\ \varrho\, [y : u = d, xs : s] &= \sigma_x\ \varrho\, [xs : s] \quad (x \neq y) \end{aligned}$$

$$\begin{aligned} \Sigma_x\ (-)\ (-) &: \mathsf{D} \times \mathsf{D} \to \mathsf{D} \\ \Sigma_x\ \varrho\, [x : u, xs : f] &= u \\ \Sigma_x\ \varrho\, [y : u, xs : f] &= \Sigma_x\ \varrho\, [xs : f(\mathsf{p}_u y\varrho)] \quad (x \neq y) \\ \Sigma_x\ \varrho\, [x : u = d, xs : s] &= u \\ \Sigma_x\ \varrho\, [y : u = d, xs : s] &= \Sigma_x\ \varrho\, [xs : s] \quad (x \neq y) \end{aligned}$$

Figure 2: Auxiliary semantic functions

$$\frac{\nu \vdash e \quad e \mid \nu = u}{\nu \vdash e : u} \qquad \nu \vdash x \qquad \frac{\nu \vdash a : \mathsf{V} \quad \forall d.\ (\nu, x : \mathsf{p_V}\, a\nu = d) \vdash e}{\nu \vdash \lambda x : a.e}$$

$$\frac{\nu \vdash a_1 : \mathsf{V} \quad \forall d.\ (\nu, x : \mathsf{p_V}\, a_1\nu = d) \vdash a_2 : \mathsf{V}}{\nu \vdash \forall x : a_1.a_2} \qquad \frac{\nu \vdash e_1 : \mathsf{Prod}\ (e_2|\nu)\ f \quad \nu \vdash e_2}{\nu \vdash e_1\ e_2}$$

$$\frac{\nu \vdash a : \mathsf{V} \quad \forall d.\ (\nu, x : \mathsf{p_V}\, a\nu = d) \vdash e : a\nu}{\nu \vdash \mu x : a.e}$$

$$\nu \vdash \{\bullet\} \qquad \frac{\nu \vdash a : \mathsf{V} \quad \nu \vdash \{d\}}{\nu \vdash \{c\ a, d\}} \qquad \frac{\nu \vdash a : \mathsf{V} \quad a\nu = \mathsf{Data}\ \Delta \quad c\Delta = \mathsf{Def}\ u \quad \nu \vdash e : u}{\nu \vdash c\ e\ a}$$

$$\frac{\nu \vdash a : \mathsf{V} \quad \nu \vdash e : \mathsf{Data}\ \Delta \quad \Delta; \nu \vdash e\nu; b : a\nu}{\nu \vdash \mathit{case}\ a\ b\ e} \qquad \Delta; \nu \vdash d; \bullet : u$$

$$\frac{\forall d'.\ (d = \mathsf{Inj}_c\ d' \implies (\nu, x : u' = d') \vdash e : u) \quad c\Delta = \mathsf{Def}\ u' \quad \Delta; \nu \vdash d; b : u}{\Delta; \nu \vdash d; (c\ x.e, b) : u}$$

$$\nu \vdash \langle \bullet \rangle \qquad \frac{\nu \vdash e \quad (\nu, x : e \mid \nu = e\nu) \vdash \langle s \rangle}{\nu \vdash \langle x = e, s \rangle} \qquad \frac{\nu \vdash a : \mathsf{V} \quad \forall d.\ (\nu, x : \mathsf{p_V}\, a\nu = d) \vdash \langle s \rangle}{\nu \vdash \langle x : a, s \rangle}$$

$$\frac{\nu \vdash e : [xs : s] \quad x \in xs}{\nu \vdash e.x} \qquad \frac{\nu \vdash a : \mathsf{V} \quad a\nu = [xs : s] \quad \nu \models r : [xs : s]}{\nu \vdash \langle r :: a \rangle}$$

$$\nu \models \bullet : [] \qquad \frac{\nu \models r : [xs : s]}{\nu \models r : [x : u = d, xs : s]} \qquad \frac{\nu \vdash e : u \quad \nu \models r : [xs : f(e\nu)]}{\nu \models (x = e, r) : [x : u, xs : f]}$$

Figure 3: Semantic type checking

notions defined in Section 3. Analogously to the semantics of expressions, we define the semantics of values and a type checking judgement for values. Sections 5–7 will then present our syntactic type checking judgement in algorithmic form.

4.1 Syntax

The syntax of values is described by the following grammar, where v are values. As well as v, we use t to denote values that are intended to represent types.

$$\begin{array}{rcl} v & := & \mathrm{v}_k^t \mid \star \mid e\rho \mid (e \mid \rho) \mid v\ v \mid t * v \\ & \mid & v{\uparrow} \mid \perp\!\!\!\perp^t \mid \gamma\ t\ b\ v\ \rho \mid \pi_c\ t \\ & \mid & \sigma_x\ v\ t \mid \Sigma_x\ v\ t \mid \sigma_x\ r\ \rho\ s\ \rho \mid \Sigma_x^{zs}\ v\ s\ \rho \\ \rho & := & () \mid (\rho, x = v) \end{array}$$

Generic values v_k^t ($k \in \mathbb{N}$) represent arbitrary values $d \in \mathsf{D}$ restricted to type t, allowing us to eliminate the universal quantifier in the semantic typing rules. The value $\star$ for the type of all types is merely convenient. In order to be able to evaluate expressions, we introduce syntactic environments ρ as sequences of pairs of identifiers and values, together with closures $e\rho$ and type closures $e \mid \rho$. The use of environments also enables us to avoid the explicit treatment of α-conversion and substitution. Application $v\ v$ and type application $t * v$ of values are introduced so that we can propagate environments to subexpressions and evaluate these separately.

Compared with Coquand and Takeyama [13], we require type inference not only for expressions but also for values $v{\uparrow}$, which explains the explicit type annotation for generic values. The type-annotated syntactic value $\perp\!\!\!\perp^t$ representing the semantic element $\perp$ will be used to integrate a simple strictness analysis into our algorithm and is necessary to establish soundness of case expressions.

The value $\gamma\ t\ b\ v\ \rho$ allows us to propagate the environment ρ of a case expression down to its type t and scrutinee v, so that we can evaluate the latter separately in order to determine the appropriate branch to select from b and the binding to perform in ρ. The value $\pi_c\ t$ is used for projecting the type associated with the constructor c out of a data type value t.

We introduce values $\sigma_x\ v\ t$ and $\Sigma_x\ v\ t$ for selection of record fields and their types, respectively. Once the subcomponents of the latter two values have been sufficiently evaluated to an actual record and/or signature, we make use of values $\sigma_x\ r\ \rho\ s\ \rho$ and $\Sigma_x^{zs}\ v\ s\ \rho$ to allow traversal of the record and/or signature in order to obtain the desired field or its type.

4.2 Semantics

The semantics of values depends on a given valuation $\delta \in \mathsf{Val} = \mathsf{D}^{Ide}$. It is defined by two continuous functions $\delta \mapsto v\delta : \mathsf{Val} \to \mathsf{D}$ and $\delta \mapsto (v \mid \delta) : \mathsf{Val} \to \mathsf{T}$. The former is the denotation of v under δ, the latter denotes the type of v under δ. Both are defined by induction on v, and given in Figure 4. Note that we overload some notation, but the

$$
\begin{array}{rlrl}
\mathrm{v}_k^t\ \delta &= \mathsf{p}_{\mathsf{p}\mathsf{V}\,t\delta}\,d, \quad \text{where } \delta(k) = d & \mathrm{v}_k^t \mid \delta &= \mathsf{p}_{\mathsf{V}}\,t\delta \\
\star\ \delta &= \mathsf{V} & \star \mid \delta &= \mathsf{V} \\
(e\rho)\ \delta &= e(\rho\delta) & (e\rho) \mid \delta &= e \mid (\rho\delta) \\
(e \mid \rho)\ \delta &= e \mid \rho\delta & (e \mid \rho) \mid \delta &= \mathsf{V} \\
v_1\ v_2\ \delta &= v_1\delta \cdot v_2\delta & v_1\ v_2 \mid \delta &= (v_1 \mid \delta) * (v_2\delta) \\
t * v\ \delta &= (t\delta) * (v\delta) & t * v \mid \delta &= \mathsf{V} \\
(v\!\uparrow)\ \delta &= v \mid \delta & (v\!\uparrow) \mid \delta &= \mathsf{V} \\
\perp\!\!\!\perp^t\ \delta &= \perp & \perp\!\!\!\perp^t \mid \delta &= \mathsf{p}_{\mathsf{V}}\,t\delta \\
\gamma\ t\ b\ v\ \rho\ \delta &= \mathsf{p}_{\mathsf{p}\mathsf{V}\,t\delta}(\llbracket b \rrbracket\ (v\delta)\ (v \mid \delta)\ \rho\delta) & \gamma\ t\ b\ v\ \rho \mid \delta &= \mathsf{p}_{\mathsf{V}}\,t\delta \\
(\pi_c\ t)\delta &= \pi_c\ (t\delta) & \pi_c\ t \mid \delta &= \mathsf{V} \\
\sigma_x\ v\ t\ \delta &= \sigma_x\ v\delta\ t\delta & \sigma_x\ v\ t \mid \delta &= \Sigma_x\ v\delta\ t\delta \\
\Sigma_x\ v\ t\ \delta &= \Sigma_x\ v\delta\ t\delta & \Sigma_x\ v\ t \mid \delta &= \mathsf{V} \\
\Sigma_x^{zs}\ v\ s\ \rho\ \delta &= \Sigma_x\ (v\delta)\ (\langle s \rangle \rho\delta) & \Sigma_x^{zs}\ v\ s\ \rho \mid \delta &= \mathsf{V} \\
\sigma_x\ r\ \rho\ s\ \rho'\ \delta &= \sigma_x\ (\tau\ (\langle s \rangle \rho'\delta)\ (r\rho\delta)) & \sigma_x\ r\ \rho\ s\ \rho' \mid \delta &= \Sigma_x\ (\tau\ (\langle s \rangle \rho'\delta)\ (r\rho\delta)) \\
& \quad (\langle s \rangle \rho'\delta) & & \quad (\langle s \rangle \rho'\delta)
\end{array}
$$

$$
\text{where} \quad \begin{array}{c} ()\delta = \perp_{\mathsf{Env}} \\ (\rho, x = v)\delta = (\rho\delta, x : v \mid \delta = v\delta) \end{array} \quad \text{and} \quad \begin{array}{c} \pi_c(-) : \mathsf{D} \to \mathsf{T} \\ \pi_c\ (\mathsf{Data}\ \Delta) = u, \quad \text{if } c\Delta = \mathsf{Def}\ u \end{array}
$$

Figure 4: Semantics of values

$$\frac{\vdash_\delta v \quad v \mid \delta = u}{\vdash_\delta v : u} \qquad \frac{\vdash_\delta t : \mathsf{V}}{\vdash_\delta \mathrm{v}_k^t} \quad \vdash_\delta \star \quad \frac{\vdash_\delta \rho \quad \rho\delta \vdash e}{\vdash_\delta e\rho} \quad \frac{\vdash_\delta e\rho}{\vdash_\delta e \mid \rho} \qquad \vdash_\delta () \quad \frac{\vdash_\delta \rho \quad \vdash_\delta v}{\vdash_\delta (\rho, x = v)}$$

$$\frac{\vdash_\delta v}{\vdash_\delta v\!\uparrow} \qquad \frac{\vdash_\delta t : \mathsf{V}}{\vdash_\delta \perp\!\!\!\perp^t} \qquad \frac{\vdash_\delta v_1 : \mathsf{Prod}\ (v_2|\delta)\ f \quad \vdash_\delta v_2}{\vdash_\delta v_1\ v_2} \qquad \frac{\vdash_\delta t \quad t\delta = \mathsf{Prod}\ (v|\delta)\ f \quad \vdash_\delta v}{\vdash_\delta t * v}$$

$$\frac{\vdash_\delta \rho \quad \vdash_\delta v : \mathsf{Data}\ \Delta \quad \vdash_\delta t : \mathsf{V} \quad \Delta; \rho\delta \vdash v\delta; b : t\delta}{\vdash_\delta \gamma\ t\ b\ v\ \rho} \qquad \frac{\vdash_\delta t \quad t\delta = \mathsf{Data}\ \Delta \quad c\Delta = \mathsf{Def}\ u}{\vdash_\delta \pi_c\ t}$$

$$\frac{\vdash_\delta t \quad \vdash_\delta v : t\delta = [xs : s] \quad x \in xs}{\vdash_\delta \sigma_x\ v\ t} \qquad \frac{\vdash_\delta t \quad \vdash_\delta v : t\delta = [xs : s] \quad x \in xs}{\vdash_\delta \Sigma_x\ v\ t}$$

$$\frac{\vdash_\delta \rho \quad \vdash_\delta \langle s\rangle\rho' \quad \rho\delta \models r : \langle s\rangle\rho'\delta \quad x \in dom(s)}{\vdash_\delta \sigma_x\ r\ \rho\ s\ \rho'}$$

$$\frac{\begin{array}{c}\vdash_\delta v : [ys : \_] \quad \vdash_\delta \langle s\rangle\rho \quad \forall z \notin zs.\ \Sigma_z\ (v\delta)\ (v \mid \delta) = \Sigma_z\ (v\delta)\ (\langle s\rangle\rho\delta) \\ x \in xs \quad zs +\!\!+ xs = ys, \text{where } \langle s\rangle\rho\delta = [xs : \_]\end{array}}{\vdash_\delta \Sigma_x^{zs}\ v\ s\ \rho}$$

Figure 5: Value type checking

distinction is clear from the context; for example, π_c is used for values but also denotes the corresponding semantic function.

Lemma 2. *For all values v and $\delta \in \mathsf{Val}$, $v\delta : v \mid \delta$.*

4.3 Value type checking

Figure 5 defines the typing judgement for values, which is based on semantic type checking of expressions. It still contains semantic notions, but is already much more syntactic in nature. The judgement $\vdash_\delta v$ says that value v is well-typed under valuation δ. Similarly to semantic type checking for expressions, it also serves as an auxiliary notion for the soundness results of our syntactic algorithm in Sections 5–7.

5 Evaluation

Figures 6–7 present the evaluation rules for values, which are of the form $v \longrightarrow\!\!\!\!\triangleright\, v'$, meaning that value v can be evaluated in one step to value v'. To limit non-termination of the inclusion algorithm in Section 6, these rules do not expand fixpoints. The additional rule for expanding fixpoints is annotated with a μ symbol over the arrow.

$$\mu x : a.e\ \rho \xrightarrow{\mu}\!\!\triangleright\ e(\rho, x = \mu x : a.e\rho)$$

When these rules are *used*, the arrow $\longrightarrow\!\!\triangleright$ denotes any evaluation rule except the fixpoint expansion rule, and the arrow $\stackrel{\mu}{\longrightarrow}\!\!\triangleright$ denotes any rule including the additional fixpoint rule.

Since the set of rules is deterministic, it defines an algorithm for applying the rules exhaustively, either excluding the fixpoint expansion rule, written $v \longrightarrow\!\!\triangleright^* v'$, or including it, written $v \stackrel{\mu}{\longrightarrow}\!\!\triangleright^* v'$.

The evaluation rules preserve the well-typing of values as well as the denotation and type denotation of well-typed values in our semantics.

Proposition 3 (Soundness of evaluation). *Let v, v' be values, and δ a valuation. If $\vdash_\delta v$ and $v \stackrel{\mu}{\longrightarrow}\!\!\triangleright v'$, then $\vdash_\delta v'$, $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$.*

Although this proposition still relies on a semantic notion of typing, later sections will establish that our syntactic type checking entails semantic type checking, so that evaluation of syntactically well-typed expressions "cannot go wrong" in the style of Milner [23].

Figure 6 contains three sets of evaluation rules, separated by dotted lines. The first set defines the lookup of identifiers and their types in environments, and explains how the elimination constructs (function application, case analysis and record field selection) for expressions relate to their value counterparts.

The second set of rules in Figure 6 defines the expected β-reduction rules for application, type application, case analysis, type projection, and record field selection. Note that reductions of application and type application forget the type of the abstracted identifier, and reduction of case analysis forgets the type annotation of the constructor; soundness of this is ensured by appropriate conditions in the typing rules in Figures 3 and 5. We need not define evaluation of type projection on empty data types or $\perp\!\!\!\perp^t$ because $\pi_c(\{\}\rho)$ and $\pi_c \perp\!\!\!\perp^t$ are ill-typed. The third set of rules in Figure 6 defines the subparts of values that need to be evaluated before attempting β-reduction.

Figure 7 contains three further sets of evaluation rules, also separated by dotted lines. The first set asserts the strictness of function application, case analysis and record field selection by defining simple symbolic computation rules over the syntactic value $\perp\!\!\!\perp^t$. This is an elegant way of integrating a strictness analysis into our algorithm in Section 6, and is crucial for establishing its soundness.

The second set of rules in Figure 7 defines the process of type inference for expressions and values. The third set is perhaps the most interesting one: it defines how case values commute with applications, type applications, type projections and record field selection. The rules are directed such that case values are moved towards the top level. We introduce auxiliary expressions $a * e$, $\pi_c\ e$, $\sigma_x\ e\ a$ and $\Sigma_x\ e\ a$ (analogous to values and defined in full in Appendix A) so that we can push type applications, type projections and record field selection into case branches; $b\ x$, $b * x$, $\pi_c\ b$, $\sigma_x\ y\ b$ and $\Sigma_x\ y\ b$ are abbreviations for mapping these constructs over branches.

We include the rules for commuting cases to the top level in order to force a specific notion of weak head normal form, inspired by the techniques of NBE and TDPE for simply-typed λ-calculus with sums [1, 7].

$$x(\rho, x = v) \longrightarrow\!\triangleright\, v \qquad\qquad \star\rho \longrightarrow\!\triangleright\, \star$$

$$x \mid (\rho, x = v) \longrightarrow\!\triangleright\, v\!\uparrow \qquad\qquad (e_1\ e_2)\ \rho \longrightarrow\!\triangleright\, (e_1\rho)\ (e_2\rho)$$

$$x(\rho, y = v) \longrightarrow\!\triangleright\, x\rho \quad (x \neq y) \qquad\qquad (case\ a\ b\ e)\ \rho \longrightarrow\!\triangleright\, \gamma\ (a\rho)\ b\ (e\rho)\ \rho$$

$$x \mid (\rho, y = v) \longrightarrow\!\triangleright\, x \mid \rho \quad (x \neq y) \qquad\qquad (e.x)\rho \longrightarrow\!\triangleright\, \sigma_x\ (e\rho)\ (e \mid \rho)$$

$$(\lambda x : a.e\ \rho)\ v \longrightarrow\!\triangleright\, e(\rho, x = v)$$

$$(\lambda x : a.e \mid \rho) * v \longrightarrow\!\triangleright\, e \mid (\rho, x = v)$$

$$(\forall x : a_1.a_2\ \rho) * v \longrightarrow\!\triangleright\, a_2(\rho, x = v)$$

$$\gamma\ t\ \bullet\ v\ \rho \longrightarrow\!\triangleright\, \perp\!\!\!\perp^t$$

$$\gamma\ t\ (c\ x.e, b)\ ((c\ e'\ a)\rho')\ \rho \longrightarrow\!\triangleright\, e(\rho, x = e'\rho')$$

$$\gamma\ t\ (c\ x.e, b)\ ((c'\ e'\ a)\rho')\ \rho \longrightarrow\!\triangleright\, \gamma\ t\ b\ ((c'\ e'\ a)\rho')\ \rho \qquad (c \neq c')$$

$$\pi_c\ (\{c\ a, d\}\rho) \longrightarrow\!\triangleright\, a\rho$$

$$\pi_c(\{c'\ a, d\}\rho) \longrightarrow\!\triangleright\, \pi_c(\{d\}\rho) \qquad (c \neq c')$$

$$\sigma_x\ (\langle r :: a\rangle\rho)\ (\langle s\rangle\rho') \longrightarrow\!\triangleright\, \sigma_x\ r\ \rho\ s\ \rho'$$

$$\Sigma_x\ v\ (\langle s\rangle\rho) \longrightarrow\!\triangleright\, \Sigma_x^{\langle\rangle}\ v\ s\ \rho$$

$$\sigma_x\ r\ \rho\ (x = e, s)\ \rho' \longrightarrow\!\triangleright\, e\rho'$$

$$\sigma_x\ (x = e, r)\ \rho\ (x : a, s)\ \rho' \longrightarrow\!\triangleright\, e\rho$$

$$\sigma_y\ r\ \rho\ (x = e, s)\ \rho' \longrightarrow\!\triangleright\, \sigma_y\ r\ \rho\ s\ (\rho', x = e\rho') \quad (y \neq x)$$

$$\sigma_y\ (x = e, r)\ \rho\ (x : a, s)\ \rho' \longrightarrow\!\triangleright\, \sigma_y\ r\ \rho\ s\ (\rho', x = e\rho) \quad (y \neq x)$$

$$\Sigma_x^{zs}\ v\ (x = e, s)\ \rho \longrightarrow\!\triangleright\, e \mid \rho$$

$$\Sigma_x^{zs}\ v\ (x : a, s)\ \rho \longrightarrow\!\triangleright\, a\rho$$

$$\Sigma_y^{zs}\ v\ (x = e, s)\ \rho \longrightarrow\!\triangleright\, \Sigma_y^{zs,x}\ v\ s\ (\rho, x = e\rho) \quad (y \neq x)$$

$$\Sigma_y^{zs}\ v\ (x : a, s)\ \rho \longrightarrow\!\triangleright\, \Sigma_y^{zs,x}\ v\ s\ (\rho, x = \sigma_x\ v\ (v\!\uparrow)) \quad (y \neq x)$$

$$\frac{v_1 \longrightarrow\!\triangleright\, v_1'}{v_1\ v_2 \longrightarrow\!\triangleright\, v_1'\ v_2} \qquad \frac{t \longrightarrow\!\triangleright\, t'}{t * v \longrightarrow\!\triangleright\, t' * v} \qquad \frac{v \longrightarrow\!\triangleright\, v'}{\gamma\ t\ (c\ x.e, b)\ v\ \rho \longrightarrow\!\triangleright\, \gamma\ t\ (c\ x.e, b)\ v'\ \rho}$$

$$\frac{t \longrightarrow\!\triangleright\, t'}{\pi_c\ t \longrightarrow\!\triangleright\, \pi_c\ t'} \qquad \frac{v \longrightarrow\!\triangleright\, v'}{\sigma_x\ v\ t \longrightarrow\!\triangleright\, \sigma_x\ v'\ t} \qquad \frac{v \not\longrightarrow\!\triangleright\, v' \quad t \longrightarrow\!\triangleright\, t'}{\sigma_x\ v\ t \longrightarrow\!\triangleright\, \sigma_x\ v\ t'} \qquad \frac{t \longrightarrow\!\triangleright\, t'}{\Sigma_x\ v\ t \longrightarrow\!\triangleright\, \Sigma_x\ v\ t'}$$

Figure 6: Evaluation rules (part 1)

$$\begin{array}{rl} \perp\!\!\!\perp^{t}\ v \longrightarrow\!\!\triangleright\ \perp\!\!\!\perp^{t*v} & \\ \gamma\ t\ (c\ x.e, b)\ \ \perp\!\!\!\perp^{v}\ \ \rho \longrightarrow\!\!\triangleright\ \perp\!\!\!\perp^{t} & \qquad \mu x : a.e\ \rho \xrightarrow{\mu}\!\!\triangleright\ e(\rho, x = \mu x : a.e\rho) \\ \sigma_x\ (\perp\!\!\!\perp^{v})\ t \longrightarrow\!\!\triangleright\ \perp\!\!\!\perp^{(\Sigma_x\ \perp\!\!\!\perp^{v}\ t)} & \end{array}$$

$$\begin{array}{rrr} \forall x : a_1.a_2 \mid \rho \longrightarrow\!\!\triangleright \star & \mu x : a.e \mid \rho \longrightarrow\!\!\triangleright a\rho & (e_1\ e_2) \mid \rho \longrightarrow\!\!\triangleright (e_1 \mid \rho) * (e_2\rho) \\ \{d\} \mid \rho \longrightarrow\!\!\triangleright \star & c\ e\ a \mid \rho \longrightarrow\!\!\triangleright a\rho & \mathit{case}\ a\ b\ e \mid \rho \longrightarrow\!\!\triangleright a\rho \\ \langle s\rangle \mid \rho \longrightarrow\!\!\triangleright \star & \langle r :: a\rangle \mid \rho \longrightarrow\!\!\triangleright a\rho & e.x \mid \rho \longrightarrow\!\!\triangleright \Sigma_x\ (e\rho)\ (e \mid \rho) \end{array}$$

$$\begin{array}{rrr} & e\rho\!\uparrow \longrightarrow\!\!\triangleright e \mid \rho & \\ \star \mid \rho \longrightarrow\!\!\triangleright \star & (e \mid \rho)\!\uparrow \longrightarrow\!\!\triangleright \star & (\sigma_x\ v\ t)\!\uparrow \longrightarrow\!\!\triangleright \Sigma_x\ v\ t \\ \mathrm{v}^t_k\!\uparrow \longrightarrow\!\!\triangleright t & (v_1\ v_2)\!\uparrow \longrightarrow\!\!\triangleright (v_1\!\uparrow) * v_2 & (\Sigma_x\ v\ t)\!\uparrow \longrightarrow\!\!\triangleright \star \\ \star\!\uparrow \longrightarrow\!\!\triangleright \star & (t * v)\!\uparrow \longrightarrow\!\!\triangleright \star & (\Sigma^{zs}_x\ v\ s\ \rho)\!\uparrow \longrightarrow\!\!\triangleright \star \\ (v\!\uparrow)\!\uparrow \longrightarrow\!\!\triangleright \star & (\gamma\ t\ b\ v\ \rho)\!\uparrow \longrightarrow\!\!\triangleright t & \dfrac{\sigma_x\ r\ \rho\ s\ \rho' \longrightarrow\!\!\triangleright v}{(\sigma_x\ r\ \rho\ s\ \rho')\!\uparrow \longrightarrow\!\!\triangleright v\!\uparrow} \\ \perp\!\!\!\perp^{t}\!\uparrow \longrightarrow\!\!\triangleright t & (\pi_c\ t)\!\uparrow \longrightarrow\!\!\triangleright \star & \end{array}$$

$$\begin{array}{l} (\gamma\ t\ b\ w\ \rho)\ v \longrightarrow\!\!\triangleright \gamma\ (t * v)\ (b\ y)\ w\ (\rho, y = v) \\ (\gamma\ t\ b\ w\ \rho) * v \longrightarrow\!\!\triangleright \gamma\ \star\ (b * y)\ w\ (\rho, y = v) \\ \pi_c(\gamma\ t\ b\ w\ \rho) \longrightarrow\!\!\triangleright \gamma\ \star\ (\pi_c\ b)\ w\ \rho \\ \sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v \longrightarrow\!\!\triangleright \gamma\ (\Sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v)\ (b.x)\ w\ \rho \\ \sigma_x\ v\ (\gamma\ t\ b\ w\ \rho) \longrightarrow\!\!\triangleright \gamma\ (\Sigma_x\ v\ (\gamma\ t\ b\ w\ \rho))\ (\sigma_x\ y\ b)\ w\ (\rho, y = v) \quad \text{(if no rule reduces } v\text{)} \\ \Sigma_x\ v\ (\gamma\ t\ b\ w\ \rho) \longrightarrow\!\!\triangleright \gamma\ \star\ (\Sigma_x\ y\ b)\ w\ (\rho, y = v) \end{array}$$

where no reduction rule applies to the value w, and y is fresh

$$\begin{array}{rl} ()\ y = () & (c\ z.e, b)\ y = (c\ z.e\ y, b\ y) \\ () * y = () & (c\ z.e, b) * y = (c\ z.e * y, b * y) \\ \pi_c\ () = () & \pi_c\ (c\ z.e, b) = (c\ z.\pi_c\ e, \pi_c\ b) \\ \sigma_x\ y\ () = () & \sigma_x\ y\ (c\ z.e, b) = (c\ z.\sigma_x\ y\ e, \sigma_x\ y\ b) \\ \Sigma_x\ y\ () = () & \Sigma_x\ y\ (c\ z.e, b) = (c\ z.\Sigma_x\ y\ e, \Sigma_x\ y\ b) \end{array}$$

Figure 7: Evaluation rules (part 2)

Values are said to be in *weak head normal form* (whnf) if no $\longrightarrow\!\!\triangleright$ evaluation rule applies, and in μ-whnf if the fixpoint rule does not apply either. *Well-typed* values in whnf are contained in the grammar h.

$$
\begin{aligned}
h &:= \star \mid \perp\!\!\!\perp^{t} \mid \lambda x : a.e\ \rho \mid (\lambda x : a.e \mid \rho) \mid \forall x : a.a\ \rho \\
&\quad \mid (c\ e\ a)\rho \mid \{d\}\rho \mid g \mid w \\
&\quad \mid \langle r :: a\rangle\rho \mid \langle s\rangle\rho \\
g &:= \gamma\ t\ b\ w\ \rho \mid \gamma\ t\ b\ g\ \rho \\
w &:= \mathrm{v}_k^t \mid (\mu x : a.e)\rho \mid w\ v \mid w * v \mid \pi_c\ w \\
&\quad \mid \Sigma_x\ v\ w \mid \sigma_x\ w\ w \mid \sigma_x\ w\ (\langle s\rangle\rho) \mid \sigma_x\ (\langle r :: a\rangle\rho)\ w
\end{aligned}
$$

If the fixpoint rule is applied, then the grammar w does not contain the value $(\mu x : a.e)\rho$. Values in the grammar h not contained in g are denoted by $\widehat{h}$. The union of grammars g and w is denoted by $\widehat{g}$.

Our inclusion algorithm in Section 6 will be guided by the structure of values in weak head normal form. Compared with Balat, Di Cosmo and Fiore [7], the grammars w, g and h are roughly equivalent to their notions of *pure neutral terms*, *neutral terms* and *pure normal terms*, respectively. Our definition of normal form is somewhat simpler than theirs because we do not specify any side conditions here; they are dealt with implicitly in our inclusion algorithm in Section 6.

6 Inclusion algorithm

This section is the heart of our technical contributions. As motivated in Section 1, we now introduce a restricted form of η-expansion to deal with commuting conversions of cases, while taking into account the subtleties of non-strict evaluation. We also extend the algorithm for subtyping recursive types by Amadio and Cardelli [3] to our language and adapt it to our semantics.

Although our type checking algorithm in Section 7 only relies on equality between values, we define a more general *inclusion* algorithm here, which then needs to be applied in both directions to obtain the desired equality. This is necessary because the soundness of our technique for recursion checking can only be established in one direction at a time in our semantic model.

First we introduce the auxiliary notions of *conversion*, *constraints* and *trails*.

6.1 Conversion

The rules of the form $v_1 \approx v_2$ given in Figure 8 define a simple conversion algorithm, which merely checks for structural equality between values.

Proposition 4 (Soundness of conversion). *Let v_1, v_2 be values, δ a valuation. If $\vdash_\delta v_i$, $v_1 \mid \delta = v_2 \mid \delta$ and $v_1 \approx v_2$, then $v_1\delta = v_2\delta$. Furthermore, if the values v_i are in the grammar w, then $\vdash_\delta w_i$ and $w_1 \approx w_2$ imply that $w_1\delta = w_2\delta$ and $w_1 \mid \delta = w_2 \mid \delta$.*

$$\frac{v_i \longrightarrow\!\!\!\triangleright^* h_i \quad h_1 \approx h_2}{v_1 \approx v_2}(v_1 \text{ or } v_2 \text{ not in whnf}) \qquad \frac{e_1\rho_1' \approx e_2\rho_2'}{(\lambda x_1 : a_1.e_1)\rho_1 \approx (\lambda x_2 : a_2.e_2)\rho_2}$$

$$\star \approx \star \qquad \frac{a_1\rho_1 \approx a_2\rho_2 \quad a_3\rho_1' \approx a_4\rho_2'}{(\forall x_1 : a_1.a_3)\rho_1 \approx (\forall x_2 : a_2.a_4)\rho_2} \qquad \frac{a_1\rho_1 \approx a_2\rho_2 \quad e_1 \mid \rho_1' \approx e_2 \mid \rho_2'}{\lambda x_1 : a_1.e_1 \mid \rho_1 \approx \lambda x_2 : a_2.e_2 \mid \rho_2}$$

$$\perp\!\!\!\perp^{t_1} \approx \perp\!\!\!\perp^{t_2} \qquad \frac{a_1\rho_1 \approx a_2\rho_2 \quad a_3\rho_1' \approx e_2 \mid \rho_2'}{(\forall x_1 : a_1.a_3)\rho_1 \approx \lambda x_2 : a_2.e_2 \mid \rho_2} \qquad \frac{a_1\rho_1 \approx a_2\rho_2 \quad e_1 \mid \rho_1' \approx a_4\rho_2'}{\lambda x_1 : a_1.e_1 \mid \rho_1 \approx (\forall x_2 : a_2.a_4)\rho_2}$$

$$\frac{t_1 \approx t_2}{\mathrm{v}_k^{t_1} \approx \mathrm{v}_k^{t_2}} \qquad \frac{w_1 \approx w_2 \quad v_1 \approx v_2}{w_1\ v_1 \approx w_2\ v_2} \qquad \frac{w_1 \approx w_2 \quad v_1 \approx v_2}{w_1 * v_1 \approx w_2 * v_2} \qquad \frac{a_1\rho_1 \approx a_2\rho_2 \quad e_1\rho_1' \approx e_2\rho_2'}{\mu x_1 : a_1.e_1\rho_1 \approx \mu x_2 : a_2.e_2\rho_2}$$

$$\frac{w_1 \approx w_2}{\pi_c\ w_1 \approx \pi_c\ w_2} \qquad \frac{dom(d_1) = dom(d_2) \quad \forall c \in dom(d_1).\ \pi_c(\{d_1\}\rho_1) \approx \pi_c(\{d_2\}\rho_2)}{\{d_1\}\rho_1 \approx \{d_2\}\rho_2}$$

$$\frac{e_1\rho_1 \approx e_2\rho_2}{(c\ e_1\ a_1)\rho_1 \approx (c\ e_2\ a_2)\rho_2}$$

$$\frac{w_1 \approx w_2 \quad w_1{\uparrow} \xrightarrow{\ \mu\ }\!\!\!\triangleright^* \{d\}\rho' \quad \{d\}\rho' \vdash \gamma\ t_1\ b_1\ w_1\ \rho_1 \approx \gamma\ t_2\ b_2\ w_2\ \rho_2}{\gamma\ t_1\ b_1\ w_1\ \rho_1 \approx \gamma\ t_2\ b_2\ w_2\ \rho_2}$$

$$\{d\}\rho' \vdash \gamma\ t_1\ \bullet\ w_1\ \rho_1 \approx \gamma\ t_2\ \bullet\ w_2\ \rho_2$$

$$\frac{e_1(\rho_1, x_1 = \mathrm{v}_k^{\pi_c(\{d\}\rho')}) \approx e_2(\rho_2, x_2 = \mathrm{v}_k^{\pi_c(\{d\}\rho')}) \quad \{d\}\rho' \vdash \gamma\ t_1\ b_1\ w_1\ \rho_1 \approx \gamma\ t_2\ b_2\ w_2\ \rho_2}{\{d\}\rho' \vdash \gamma\ t_1\ (c\ x_1.e_1, b_1)\ w_1\ \rho_1 \approx \gamma\ t_2\ (c\ x_2.e_2, b_2)\ w_2\ \rho_2}$$

$$\langle\bullet\rangle\rho_1 \approx \langle\bullet\rangle\rho_2 \qquad \frac{a_1\rho_1 \approx a_2\rho_2 \quad \langle s_1\rangle(\rho_1, x = \mathrm{v}_k^{a_1\rho_1}) \approx \langle s_2\rangle(\rho_2, x = \mathrm{v}_k^{a_2\rho_2})}{\langle x : a_1, s_1\rangle\rho_1 \approx \langle x : a_2, s_2\rangle\rho_2}$$

$$\frac{e_1 \mid \rho_1 \approx e_2 \mid \rho_2 \quad e_1\rho_1 \approx e_2\rho_2 \quad \langle s_1\rangle(\rho_1, x = e_1\rho_1) \approx \langle s_2\rangle(\rho_2, x = e_2\rho_2)}{\langle x = e_1, s_1\rangle\rho_1 \approx \langle x = e_2, s_2\rangle\rho_2}$$

$$\frac{r_1\rho_1 \sim r_2\rho_2}{\langle r_1 :: a_1\rangle\rho_1 \approx \langle r_2 :: a_2\rangle\rho_2} \qquad \bullet\rho_1 \sim \bullet\rho_2 \qquad \frac{e_1\rho_1 \approx e_2\rho_2 \quad r_1\rho_1 \sim r_2\rho_2}{(x = e_1, r_1)\rho_1 \sim (x = e_2, r_2)\rho_2}$$

$$\frac{v_1 \approx v_2 \quad t_1 \approx t_2}{\sigma_x\ v_1\ t_1 \approx \sigma_x\ v_2\ t_2} \qquad \frac{v_1 \approx v_2 \quad t_1 \approx t_2}{\Sigma_x\ v_1\ t_1 \approx \Sigma_x\ v_2\ t_2}$$

where $\rho_i' = (\rho_i, x_i = \mathrm{v}_k^{a_i\rho_i})$ and $\mathrm{v}_k^{\cdot\cdot}$ not occurring in the respective rule's conclusion

Figure 8: Conversion algorithm

6.2 Constraints

The notion of constraints is inspired by the NBE and TDPE techniques for typed λ-terms with sums [1, 7].

Constraints are assertions of equalities between values, and will be introduced as assumptions when decomposing case expressions. Syntactically, constraints are lists of pairs of values, where the left-hand value is contained in the grammar w.

$$\theta \quad := \quad () \mid (\theta, w \approx v)$$

Our well-typing condition for constraints requires that all pairs of values are well-typed and their value and type denotations are equal in our semantic model.

$$\vdash_\delta () \qquad \frac{\vdash_\delta \theta \quad \vdash_\delta w \quad \vdash_\delta v \quad w\delta = v\delta \quad w \mid \delta = v \mid \delta}{\vdash_\delta (\theta, w \approx v)}$$

Both sides of a constraint are expected to belong to a data type, but we need not add such a condition here, as we will be able to infer it from the context when required. The right-hand value of a constraint will always be a constructor of the following form.

$$(c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho)}, y = \{d\}\rho)$$

We define a family ($\longrightarrow\!\!\triangleright_\theta$) of additional evaluation rules for applying constraints in θ, which says that a value in the grammar w that is convertible to the left-hand value of a constraint $(w' \approx v) \in \theta$ evaluates to the right-hand value v.

$$\frac{(w' \approx v) \in \theta \qquad w \approx w'}{w \longrightarrow\!\!\triangleright_\theta v}$$

If the fixpoint rule ($\xrightarrow{\mu}\!\!\triangleright$) is not used, our set of evaluation rules remains deterministic because no other rule can reduce values in the grammar w; otherwise we expand fixpoints before applying constraints. Exhaustive application of evaluation rules including application of constraints in θ is written $v \longrightarrow\!\!\triangleright_\theta^* h$ or $v \xrightarrow{\mu}\!\!\triangleright_\theta^* h$, where h is said to be in θ-whnf or θ-μ-whnf, respectively.

We generalise Proposition 3 to include the constraint application rule. Note that the well-typing condition for constraints in the premise ensures that applying constraints merely involves substituting equals for equals.

Proposition 5 (Soundness of evaluation). *Let v, v' be values, θ constraints and δ a valuation. If $\vdash_\delta \theta$, $\vdash_\delta v$ and $v \xrightarrow{\mu}\!\!\triangleright_\theta v'$, then $\vdash_\delta v'$, $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$.*

6.3 Trails

The notion of trails is inspired by Amadio and Cardelli's algorithm for subtyping recursive types [3]. Intuitively, trails consist of the pairs of values that were encountered immediately before expanding a (guarded) fixpoint during inclusion checking. Roughly, the idea is that if we encounter the same pair of values again without having found any inconsistencies, then repeating the same fixpoint expansion cannot reveal any inconsistencies either, so their infinite expansions must be included in each other.

Syntactically, our trails are pairs of values, where the left-hand value is in the grammar $\underline{w} := \mathrm{v}_k^t \mid \underline{w}\ v$.

$$\phi \quad := \quad () \mid (\phi, \underline{w} \preccurlyeq v)$$

This definition of trails differs from Amadio and Cardelli's due to our choice of semantics. Instead of placing actual fixpoints into our trail, we record generic values as representatives, and only if the fixpoint occurred on the *left-hand side* of the inclusion. In addition, we record *applications* of fixpoints to other values, thus extending Amadio and Cardelli's algorithm to the use of functions and function applications within types.

Our well-typing condition for trails requires each pair of values in a trail to be well-typed, and their value and type denotations to be included in each other.

$$\vdash_\delta () \qquad \frac{\vdash_\delta \phi \quad \vdash_\delta \underline{w} \quad \vdash_\delta v \quad \underline{w}\delta \sqsubseteq v\delta \quad \underline{w} \mid \delta \sqsubseteq v \mid \delta}{\vdash_\delta (\phi, \underline{w} \preccurlyeq v)}$$

6.4 Inclusion

Figures 9–11 present our inclusion algorithm using rules of the form $\phi; \theta \vdash v_1 \preccurlyeq v_2$, meaning that value v_1 is included in value v_2 under trail ϕ and constraints θ. The soundness of our inclusion rules is given as follows.

Proposition 6 (Soundness of inclusion). *Let ϕ be a trail, θ constraints, v_1, v_2 values and δ a valuation. If $\vdash_\delta \phi$, $\vdash_\delta \theta$, $\vdash_\delta v_i$, $v_1 \mid \delta \sqsubseteq v_2 \mid \delta$ and $\phi; \theta \vdash v_1 \preccurlyeq v_2$, then $v_1\delta \sqsubseteq v_2\delta$.*

If our inclusion rules assert the inclusion of two values, then their denotations are included in each other in our semantic model under the above conditions; these will be established by our syntactic type checking algorithm in order to make use of the inclusion algorithm.

The first rule in Figure 9 says that if either of the two values is not in θ-whnf, it should be reduced to it, using constraints in θ but *without* expanding fixpoints; all other rules in Figures 9–11 are only used if both values are in θ-whnf. The remaining rules in Figure 9 simply check for structural inclusion.

Figure 10 contains η-rules for functions, records and cases. If one of the two values is an abstraction and the other an undetermined value in the grammar w, we perform an η-expansion of w. This is followed immediately by a structural inclusion check, which is incorporated into the rule. We now consider the η-rules for cases in Figure 10, and the rules for fixpoints in Figure 11.

6.4.1 Case values

To check whether a case value g is included in a value h, we first infer the type of the case value's *innermost* scrutinee $w = \textsc{scrutinee}(g)$ and verify that it evaluates to some

$$\frac{v_i \longrightarrow^*_\theta h_i \quad \phi;\theta \vdash h_1 \preccurlyeq h_2}{\phi;\theta \vdash v_1 \preccurlyeq v_2}(v_1 \text{ or } v_2 \text{ not in } \theta\text{-whnf})$$

$$\phi;\theta \vdash \star \preccurlyeq \star \qquad \phi;\theta \vdash \perp\!\!\!\perp^t \preccurlyeq h \qquad \frac{\phi;\theta \vdash e_1\rho_1' \preccurlyeq e_2\rho_2'}{\phi;\theta \vdash \lambda x_1 : a_1.e_1\rho_1 \preccurlyeq \lambda x_2 : a_2.e_2\rho_2}$$

$$\frac{\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2 \quad \phi;\theta \vdash a_3\rho_1' \preccurlyeq a_4\rho_2'}{\phi;\theta \vdash \forall x_1 : a_1.a_3\ \rho_1 \preccurlyeq \forall x_2 : a_2.a_4\ \rho_2} \qquad \frac{\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2 \quad \phi;\theta \vdash e_1 \mid \rho_1' \preccurlyeq e_2 \mid \rho_2'}{\phi;\theta \vdash \lambda x_1 : a_1.e_1 \mid \rho_1 \preccurlyeq \lambda x_2 : a_2.e_2 \mid \rho_2}$$

$$\frac{\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2 \quad \phi;\theta \vdash a_3\rho_1' \preccurlyeq e_2 \mid \rho_2'}{\phi;\theta \vdash \forall x_1 : a_1.a_3\ \rho_1 \preccurlyeq \lambda x_2 : a_2.e_2 \mid \rho_2} \qquad \frac{\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2 \quad \phi;\theta \vdash e_1 \mid \rho_1' \preccurlyeq a_4\rho_2'}{\phi;\theta \vdash \lambda x_1 : a_1.e_1 \mid \rho_1 \preccurlyeq \forall x_2 : a_2.a_4\ \rho_2}$$

$$\frac{\phi;\theta \vdash e_1\rho_1 \preccurlyeq e_2\rho_2}{\phi;\theta \vdash (c\ e_1\ a_1)\rho_1 \preccurlyeq (c\ e_2\ a_2)\rho_2}$$

$$\frac{dom(d_1) \subseteq dom(d_2) \quad \forall c \in dom(d_1).\ \phi;\theta \vdash \pi_c(\{d_1\}\rho_1) \preccurlyeq \pi_c(\{d_2\}\rho_2)}{\phi;\theta \vdash \{d_1\}\rho_1 \preccurlyeq \{d_2\}\rho_2}$$

$$\phi;\theta \vdash \langle\bullet\rangle\rho_1 \preccurlyeq \langle\bullet\rangle\rho_2 \qquad \frac{\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2 \quad \phi;\theta \vdash \langle s_1\rangle(\rho_1, x = \mathrm{v}_k^{a_1\rho_1}) \preccurlyeq \langle s_2\rangle(\rho_2, x = \mathrm{v}_k^{a_2\rho_2})}{\phi;\theta \vdash \langle x : a_1, s_1\rangle\rho_1 \preccurlyeq \langle x : a_2, s_2\rangle\rho_2}$$

$$\frac{\phi;\theta \vdash e_1 \mid \rho_1 \preccurlyeq e_2 \mid \rho_2 \quad \phi;\theta \vdash e_1\rho_1 \preccurlyeq e_2\rho_2 \quad \phi;\theta \vdash \langle s_1\rangle(\rho_1, x = e_1\rho_1) \preccurlyeq \langle s_2\rangle(\rho_2, x = e_2\rho_2)}{\phi;\theta \vdash \langle x = e_1, s_1\rangle\rho_1 \preccurlyeq \langle x = e_2, s_2\rangle\rho_2}$$

$$\frac{\phi;\theta \models r_1\rho_1 \preccurlyeq r_2\rho_2}{\phi;\theta \vdash \langle r_1 :: a_1\rangle\rho_1 \preccurlyeq \langle r_2 :: a_2\rangle\rho_2}$$

$$\phi;\theta \models \bullet\rho_1 \preccurlyeq \bullet\rho_2 \qquad \frac{\phi;\theta \vdash e_1\rho_1 \preccurlyeq e_2\rho_2 \quad \phi;\theta \models r_1\rho_1 \preccurlyeq r_2\rho_2}{\phi;\theta \models (x = e_1, r_1)\rho_1 \preccurlyeq (x = e_2, r_2)\rho_2}$$

$$\frac{\phi;\theta \vdash t_1 \preccurlyeq t_2}{\phi;\theta \vdash \mathrm{v}_k^{t_1} \preccurlyeq \mathrm{v}_k^{t_2}} \qquad \frac{\phi;\theta \vdash w_1 \preccurlyeq w_2 \quad \phi;\theta \vdash v_1 \preccurlyeq v_2}{\phi;\theta \vdash w_1\ v_1 \preccurlyeq w_2\ v_2} \qquad \frac{\phi;\theta \vdash w_1 \preccurlyeq w_2 \quad \phi;\theta \vdash v_1 \preccurlyeq v_2}{\phi;\theta \vdash w_1 * v_1 \preccurlyeq w_2 * v_2}$$

$$\frac{\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2 \quad \phi;\theta \vdash e_1\rho_1' \preccurlyeq e_2\rho_2'}{\phi;\theta \vdash \mu x_1 : a_1.e_1\ \rho_1 \preccurlyeq \mu x_2 : a_2.e_2\ \rho_2} \qquad \frac{\phi;\theta \vdash w_1 \preccurlyeq w_2}{\phi;\theta \vdash \pi_c\ w_1 \preccurlyeq \pi_c\ w_2}$$

$$\frac{\phi;\theta \vdash v_1 \preccurlyeq v_2 \quad \phi;\theta \vdash t_1 \preccurlyeq t_2}{\phi;\theta \vdash \sigma_x\ v_1\ t_1 \preccurlyeq \sigma_x\ v_2\ t_2} \qquad \frac{\phi;\theta \vdash v_1 \preccurlyeq v_2 \quad \phi;\theta \vdash t_1 \preccurlyeq t_2}{\phi;\theta \vdash \Sigma_x\ v_1\ t_1 \preccurlyeq \Sigma_x\ v_2\ t_2}$$

where $\rho_i' = (\rho_i, x_i = \mathrm{v}_k^{a_i\rho_i})$ with $\mathrm{v}_k^{\cdot\cdot}$ not occurring in the respective rule's conclusion

Figure 9: Inclusion algorithm (structural rules)

$$\frac{[w\uparrow \xrightarrow{\mu}\!\!\triangleright^*_\theta \lambda x_2 : a_2.e_2 \mid \rho_2 \quad \text{or} \xrightarrow{\mu}\!\!\triangleright^*_\theta \forall x_2 : a_2.a_4\ \rho_2] \qquad \phi;\theta \vdash e_1\rho_1' \preccurlyeq w\ \mathrm{v}_k^{a_2\rho_2}}{\phi;\theta \vdash \lambda x_1 : a_1.e_1\rho_1 \preccurlyeq w}$$

$$\frac{[w\uparrow \xrightarrow{\mu}\!\!\triangleright^*_\theta \lambda x_1 : a_1.e_1 \mid \rho_1 \quad \text{or} \xrightarrow{\mu}\!\!\triangleright^*_\theta \forall x_1 : a_1.a_3\ \rho_1] \qquad \phi;\theta \vdash w\ \mathrm{v}_k^{a_1\rho_1} \preccurlyeq e_2\rho_2'}{\phi;\theta \vdash w \preccurlyeq \lambda x_2 : a_2.e_2\rho_2}$$

$$\frac{a\rho \xrightarrow{\mu}\!\!\triangleright^*_\theta \langle s\rangle\rho' \quad \forall x \in dom(s).\ \phi;\theta \vdash \sigma_x\ (\langle r :: a\rangle\rho)\ (a\rho) \preccurlyeq \sigma_x\ w\ (w\uparrow)}{\phi;\theta \vdash \langle r :: a\rangle\rho \preccurlyeq w}$$

$$\frac{w\uparrow \xrightarrow{\mu}\!\!\triangleright^*_\theta \langle s\rangle\rho' \quad \forall x \in dom(s).\ \phi;\theta \vdash \sigma_x\ w\ (w\uparrow) \preccurlyeq \sigma_x\ (\langle r :: a\rangle\rho)\ (a\rho)}{\phi;\theta \vdash w \preccurlyeq \langle r :: a\rangle\rho}$$

where $\rho_i' = (\rho_i, x_i = \mathrm{v}_k^{a_i\rho_i})$ with $\mathrm{v}_k^{\cdot\cdot}$ not occurring in the respective rule's conclusion

. . . .

$$\frac{w\uparrow \xrightarrow{\mu}\!\!\triangleright^*_\theta \{d\}\rho \quad \forall c \in dom(d).\ \phi;\theta_c \vdash g \preccurlyeq h}{\phi;\theta \vdash g \preccurlyeq h}$$

$$\frac{\phi;(\theta, w \approx \perp\!\!\!\perp^{(w\uparrow)}) \vdash \widehat{h} \preccurlyeq \perp\!\!\!\perp^{(g\uparrow)} \quad w\uparrow \xrightarrow{\mu}\!\!\triangleright^*_\theta \{d\}\rho \quad \forall c \in dom(d).\ \phi;\theta_c \vdash \widehat{h} \preccurlyeq g}{\phi;\theta \vdash \widehat{h} \preccurlyeq g}$$

where
$$w = \text{SCRUTINEE}\ (g)$$
$$\theta_c = (\theta, w \approx (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho)}, y = \{d\}\rho))$$

and $\mathrm{v}_k^{\cdot\cdot}$ does not occur in the above rules' conclusions

and
$$\boxed{\begin{array}{l}\text{SCRUTINEE}(\gamma\ t\ b\ g\ \rho) = \text{SCRUTINEE}(g)\\ \text{SCRUTINEE}(\gamma\ t\ b\ w\ \rho) = w\end{array}}$$

Figure 10: Inclusion algorithm (η-rules for functions, records and cases)

$$\frac{(\underline{w}_2 \preccurlyeq h_2) \in \phi \qquad \underline{w}_1 \approx \underline{w}_2 \qquad h_1{\uparrow} \approx h_2{\uparrow} \qquad h_i \xrightarrow{\mu}\!\!\triangleright^*_\theta h'_i \qquad h'_1 \approx h'_2}{\phi;\theta \vdash \underline{w}_1 \preccurlyeq h_1}(2)$$

$$\forall n \in \mathbb{N}.\ \frac{\phi;\theta \vdash \overline{w}{\uparrow} \preccurlyeq h{\uparrow} \quad (\phi, \mathrm{G}(\mathrm{v}_k^{a\rho}) \preccurlyeq h);\theta \vdash \mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho})) \preccurlyeq h}{\phi;\theta \vdash \overline{w} \preccurlyeq h}(3a)$$

$$\text{where} \quad \begin{array}{r} \mu x : a.e\rho = \mathrm{BASE}(\overline{w}) \\ \mathrm{G} = \mathrm{STEP}(\overline{w}) \end{array} \quad \text{and} \quad \begin{array}{r} \mathrm{F}^0\ v = e(\rho, x = v) \\ \mathrm{F}^{i+1}\ v = \mathrm{F}^0(\mathrm{F}^i\ v) \end{array} \quad \text{and} \quad \begin{array}{l} \mathrm{v}_k^{\because} \text{ does not occur in} \\ \text{this rule's conclusion} \end{array}$$

$$\text{and} \quad \boxed{\begin{array}{ll} \mathrm{BASE}(\mu x : a.e\rho) = \mu x : a.e\rho & \mathrm{STEP}(\mu x : a.e\rho)\ v = v \\ \mathrm{BASE}(\overline{w}\ v) = \mathrm{BASE}(\overline{w}) & \mathrm{STEP}(\overline{w}\ v)\ v' = (\mathrm{STEP}(\overline{w})\ v')\ v \end{array}}$$

$$\frac{\phi;\theta \vdash w{\uparrow} \preccurlyeq h{\uparrow} \quad w \xrightarrow{\mu}\!\!\triangleright^*_\theta h_1 \quad h \xrightarrow{\mu}\!\!\triangleright^*_\theta h_2 \quad \phi;\theta \vdash h_1 \preccurlyeq h_2}{\phi;\theta \vdash w \preccurlyeq h}(3b)(w \text{ not in } \theta\text{-}\mu\text{-whnf})$$

$$\frac{\phi;\theta \vdash h{\uparrow} \preccurlyeq w{\uparrow} \quad \phi;\theta \vdash w \preccurlyeq \perp\!\!\!\perp^{(w\uparrow)} \quad \phi;\theta \vdash h \preccurlyeq \perp\!\!\!\perp^{(w\uparrow)}}{\phi;\theta \vdash h \preccurlyeq w}(3c)(h \text{ in } \theta\text{-}\mu\text{-whnf, but not } w)$$

$$\frac{\phi;\theta \vdash h_1{\uparrow} \preccurlyeq w{\uparrow} \quad w \xrightarrow{\mu}\!\!\triangleright^*_\theta h_2 \quad \phi;\theta \vdash h_1 \preccurlyeq h_2}{\phi;\theta \vdash h_1 \preccurlyeq w}(3c)(h_1 \text{ in } \theta\text{-}\mu\text{-whnf, but not } w)$$

Figure 11: Inclusion algorithm (fixpoint rules)

data type $\{d\}\rho$. We then repeat the inclusion test for each constructor c declared in d, using a constraint

$$(w \approx (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho)}, y = \{d\}\rho))$$

that is added to θ. Essentially, this can be viewed as an η-expansion of the innermost scrutinee of g, followed immediately by a decomposition into each branch. The additional constraint is kept track of globally within the premises of this rule, and can be "plugged in" on either side of the inclusion check using the evaluation rule for constraint application. This is how our algorithm verifies inclusions involving commuting conversions of cases.

Another way of viewing the η-rule for cases is that we "split" the scrutinee w into all its possible values, that is, all constructors c in the data type definition d, and verify the inclusion for each c separately. This is in fact how our proof proceeds, and it is the only place in our algorithm where we need to deal syntactically with undefined values: the scrutinee w could be undefined, in which case g would also be undefined. No problem arises while verifying inclusion of g in h; for the other direction, we add the constraint $(\theta, w \approx \perp\!\!\!\perp^{(w\uparrow)})$ to θ.

We invite the reader to check that this simple technique incorporates the necessary strictness analysis for the distribution rule for cases introduced in Section 1.

$$f(\mathit{case}\ e\ \mathit{of} \ldots b_i \ldots) \;=\; \mathit{case}\ e\ \mathit{of} \ldots (f\ b_i) \ldots$$

Analogous to the conditions for constraints specified in NBE for simply-typed λ-calculus with sums [1], we maintain that: (i) no two distinct constraints involve the same term w (modulo conversion) on the left-hand side, because g is in θ-whnf and its scrutinee w can therefore not occur in θ, and (ii) no two distinct constraints involve the same generic value on the right-hand side because of the freshness condition for generic values.

As part of ensuring uniqueness of normal forms, TDPE for typed λ-terms with sums [7] imposes three side conditions on the structure of their normal forms: (A) the relative positions of cases and abstractions are fixed, (B) dead branches, which arise from nested case analyses on the same scrutinee, are not allowed and (C) the branches of a case analysis cannot all be the same because such a case analysis would be redundant.

These conditions are dealt with implicitly by our inclusion algorithm, because (A) constraints are generated and plugged in "on demand", that is whenever we encounter a case analysis on either side of the inclusion, (B) dead branches are ignored by plugging in constraints several times, and (C) duplicate branches do not need to be checked against each other, because they are checked against the other side of the inclusion.

6.4.2 Fixpoints

The inclusion rules in the first two sets are mutually exclusive. We now consider the third set, containing the overlapping rules (2) and (3a–d) for fixpoints, which are applied if the naive structural rule in the first set fails.

First, rule (2) determines whether the two values being checked are contained in the trail ϕ (modulo conversion), in which case the inclusion check succeeds.

Otherwise, we proceed as follows. If the left-hand value is in the grammar $\overline{w} := \mu x : a.e\rho \mid \overline{w}\ v$, rules (3a) and (3b) apply. If the left-hand value is a fixpoint expression w not in the grammar $\overline{w}$, rule (3b) applies. Otherwise, if the right-hand value involves a fixpoint expression in the grammar w, rules (3c) and (3d) apply.

Rule (3a) says that in order to check whether a fixpoint $\overline{w} = \textsc{g}(\mu x : a.e\rho)$ is included in a value h, we add the pair of values $(\textsc{g}(\mathrm{v}_k^{a\rho}) \preccurlyeq h)$ to the trail ϕ, and expand the fixpoint n times to obtain $\textsc{g}(\textsc{f}^n(\mu x : a.e\rho))$ on the left-hand side of the inclusion, but we replace the fixpoint $\mu x : a.e\rho$ in the latter by $\mathrm{v}_k^{a\rho}$. Soundness of this rule is justified by fixpoint induction over the predicate $(\textsc{g}\delta)\ d \sqsubseteq h\delta$, which is admissible (chain-complete) in $d \in \mathsf{D}$, and where $\textsc{g}\delta$ is the evident semantic analogue to the syntactic function $\textsc{g}$. This explains the need for our more general *inclusion* algorithm: the base step of the fixpoint induction proof only holds in one direction. Our rule (3a) is a generalised version of the analogous rule in Amadio and Cardelli [3], where $\textsc{g}$ is the identity.

Our algorithm applies the rule (3a) in the following order. First we apply it for $n = 0$. If at any step $n = k$ the rule succeeds, then we are done. If the only inconsistencies we find are caused by rule (2), where the left-hand value matches a left-hand value in the trail but the right-hand values do not match, then expanding the fixpoint k times is not sufficient for verifying the inclusion and we need to attempt rule (3a) for $n = k + 1$. Any other inconsistencies cause the rule (3a) to fail, in which case we proceed by expanding fixpoints on both sides of the inclusion using rule (3b).

As explained above, the strategy of rule (3a) cannot be used for fixpoints occurring on the right-hand side, so we simply unroll them using rule (3d). This is not problematic if the right-hand side w is a *guarded* fixpoint, because we will eventually either reach a fixpoint on the left-hand side, so that we can use rule (3a), or find an inconsistency. To limit the divergence of rule (3d) if the right-hand side w is an *unguarded* fixpoint, we first try to find a derivation using rule (3c) to show that w is undefined by moving it to the left-hand side: if the inclusion $\phi; \theta \vdash w \preccurlyeq \perp\!\!\!\perp^{w\uparrow}$ succeeds, then we can replace w with the symbolic value $\perp\!\!\!\perp^{w\uparrow}$ and proceed by checking $\phi; \theta \vdash h \preccurlyeq \perp\!\!\!\perp^{w\uparrow}$ instead of using rule (3d). Our rule (3c) replaces the explicit conversion of terms to *canonical form* in Amadio and Cardelli [3].

We invite the reader to verify how our inclusion algorithm succeeds in both directions on the list types given in Section 1, which involve functions and their application, and where the fixpoints are "out of step".

$$\begin{aligned} list_1 &= \mu\ l : \star \to \star\ .\ \lambda a : \star\ .\ \{\mathsf{Nil},\ \mathsf{Cons}\ a\ (l\ a)\} \\ list_2 &= \lambda a : \star\ .\ \mu\ l : \star\ .\ \{\mathsf{Nil},\ \mathsf{Cons}\ a\ l\} \end{aligned}$$

Note that multiple constructor arguments can be encoded using our dependent records.

6.4.3 Limitations

We have shown how to adapt key ideas from NBE and TDPE for simply-typed λ-calculus with sums [1, 7] to our setting in order to deal with commuting conversions of case expressions under lazy evaluation: we exploit the structure of their normal forms, adopt the notion of constraints and integrate a restricted notion of η-reduction based on the structure of *terms* into our algorithm. Integrating the full power of NBE and TDPE based on an η-expansion guided by the structure of types is out of the scope of this paper and remains further work. We expect the presence of recursive types in our setting to cause undesirable non-termination, but it would be interesting to see to what extent these techniques can be integrated into our algorithm for non-recursive types.

Our extension of the algorithm by Amadio and Cardelli [3] to take into account functions is necessarily rather limited, as extensional equality of functions is undecidable. Since the algorithm relies on finding values in the trail that have already been encountered, it only succeeds for functions whose recursive calls involve the same argument that the function was originally applied to; these can be identified with equivalent definitions involving a non-recursive argument. In other words, our extension can "commute fixpoints through abstractions" if this transformation is valid.

Other equalities between functions require the use of inclusion rule (3b), which causes non-termination due to η-expansion when checking semantically valid inclusions whose verification requires the use of an inductive proof.

For example, consider a recursively defined type *nat* of natural numbers and an addition function *plus* defined by case analysis on its *first* argument.

$$\begin{aligned} nat &= \mu\, n : \star.\ \{\mathsf{O},\ \mathsf{S}\ n\} \\ plus &= \mu\, plus : (nat \to nat \to nat).\ \lambda n : nat.\ \lambda m : nat. \\ &\qquad case\ nat\ (\mathsf{O}.\ m,\ \mathsf{S}\ k.\ \mathsf{S}\ (plus\ k\ m))\ n \end{aligned}$$

Non-termination arises when verifying inclusion of the function $oneplus_1$ in $oneplus_2$, defined as follows.

$$oneplus_1 = \lambda n : nat.\lambda m : nat.\ plus\ n\ (\mathsf{S}\ m)$$
$$oneplus_2 = \lambda n : nat.\lambda m : nat.\ \mathsf{S}\ (plus\ n\ m)$$

After binding the arguments n and m to fresh generic values, say v_1^{nat} and v_0^{nat}, the inclusion algorithm enters the following cycle. To check the inclusion

$$plus\ (\mathrm{v}_1^{nat})\ (\mathsf{S}\ \mathrm{v}_0^{nat}) \preccurlyeq \mathsf{S}\ (plus\ (\mathrm{v}_1^{nat})\ \mathrm{v}_0^{nat})$$

it expands the fixpoint in the definition of *plus* on the left-hand side, generates constraints for the scrutinee v_1^{nat}, and proceeds to check the following two inclusions.

$$plus\ \mathsf{O}\ (\mathsf{S}\ \mathrm{v}_0^{nat}) \preccurlyeq \mathsf{S}\ (plus\ \mathsf{O}\ \mathrm{v}_0^{nat})$$
$$plus\ (\mathsf{S}\ \mathrm{v}_2^{nat})\ (\mathsf{S}\ \mathrm{v}_0^{nat}) \preccurlyeq \mathsf{S}\ (plus\ (\mathsf{S}\ \mathrm{v}_2^{nat})\ \mathrm{v}_0^{nat})$$

The first succeeds by definition of *plus* and structural inclusion, but the second only reduces to the following.

$$plus\ (\mathrm{v}_2^{nat})\ (\mathsf{S}\ \mathrm{v}_0^{nat}) \preccurlyeq \mathsf{S}\ (plus\ (\mathrm{v}_2^{nat})\ \mathrm{v}_0^{nat})$$

This is the inclusion we started with, except that one of the generic values is replaced with a fresh one. Since it is semantically valid, we never find an inconsistency and the algorithm diverges essentially by trying to generate all possible input values of n. It may be possible to detect this behaviour to some extent using circularity checks, but ultimately we cannot see any solution other than reporting to the programmer that the number of recursive function expansions (or generated fresh values) has exceeded a specified limit.

Another cause of non-termination is that our evaluation rules are not strongly normalising, but we believe this to be less of a practical concern. Because the same problem occurs at program run-time, programmers wishing to use general recursion at that stage should see no reason to object to it if they make type-level computations dependent on it.

7 Syntactic type checking

Having established all auxiliary notions, we are now in a position to state our syntactic type checking algorithm in Figure 12. The judgement $\theta; \rho \vdash e$ says that expression e is well-typed under syntactic environment ρ and constraints θ. Notice that the rule $\theta; \rho \vdash e : t$ verifies that the type of e is equal to t by applying the inclusion algorithm in both directions.

The rule for case expressions is the most intricate one. It involves the auxiliary judgement $\theta; \rho \vdash h; b : a$, which verifies that the branches b have type a under constraints θ and environment ρ, and where h is the value being scrutinised by the case analysis. An invariant of this judgement is that h is in θ-whnf and belongs to a data type. It

$$\frac{\theta;\rho\vdash e \quad ();\theta\vdash e\mid\rho\preccurlyeq t \quad ();\theta\vdash t\preccurlyeq e\mid\rho}{\theta;\rho\vdash e : t} \qquad \theta;\rho\vdash\star \qquad \frac{x\in\mathrm{dom}\ \rho}{\theta;\rho\vdash x}$$

$$\frac{\theta;\rho\vdash a:\star \quad \theta;(\rho,x=\mathrm{v}_k^{a\rho})\vdash e}{\theta;\rho\vdash\lambda x:a.e} \qquad \frac{\theta;\rho\vdash a_1:\star \quad \theta;(\rho,x=\mathrm{v}_k^{a_1\rho})\vdash a_2:\star}{\theta;\rho\vdash\forall x:a_1.a_2}$$

$$\frac{\theta;\rho\vdash a:\star \quad \theta;(\rho,x=\mathrm{v}_k^{a\rho})\vdash e : a\rho}{\theta;\rho\vdash\mu x:a.e} \qquad \frac{\begin{bmatrix} e_1\mid\rho \xrightarrow{\mu}\!\!\triangleright_\theta^* \lambda x:a.e\mid\rho' \\ \text{or} \xrightarrow{\mu}\!\!\triangleright_\theta^* \forall x:a.a_2\rho' \end{bmatrix} \quad \begin{matrix}\theta;\rho\vdash e_1 \\ \theta;\rho\vdash e_2 : a\rho'\end{matrix}}{\theta;\rho\vdash e_1\ e_2}$$

$$\theta;\rho\vdash\{\bullet\} \qquad \frac{\theta;\rho\vdash a:\star \quad \theta;\rho\vdash\{d\}}{\theta;\rho\vdash\{c\ a, d\}}$$

$$\frac{\theta;\rho\vdash a:\star \quad a\rho\xrightarrow{\mu}\!\!\triangleright_\theta^* \{d\}\rho' \quad \theta;\rho\vdash e:\pi_c(\{d\}\rho') \quad c\in dom(d)}{\theta;\rho\vdash c\ e\ a}$$

$$\frac{\begin{matrix}\theta;\rho\vdash e \quad e\mid\rho\xrightarrow{\mu}\!\!\triangleright_\theta^* \{d\}\rho' \quad e\rho \longrightarrow\!\!\triangleright_\theta^* h \\ \theta;\rho\vdash a:\star \quad \theta;\rho\vdash h;b:a \quad \forall c\in dom(b).\ c\in dom(d)\end{matrix}}{\theta;\rho\vdash case\ a\ b\ e}$$

$$\theta;\rho\vdash\langle\bullet\rangle \qquad \frac{\theta;\rho\vdash e \quad \theta;(\rho,x=e\rho)\vdash\langle s\rangle}{\theta;\rho\vdash\langle x=e,s\rangle} \qquad \frac{\theta;\rho\vdash a:\star \quad \theta;(\rho,x=\mathrm{v}_k^{a\rho})\vdash\langle s\rangle}{\theta;\rho\vdash\langle x:a,s\rangle}$$

$$\frac{\theta;\rho\vdash e \quad e\mid\rho\xrightarrow{\mu}\!\!\triangleright_\theta^* \langle s\rangle\rho' \quad x\in dom(s)}{\theta;\rho\vdash e.x} \qquad \frac{\theta;\rho\vdash a:\star \quad a\rho\xrightarrow{\mu}\!\!\triangleright_\theta^*\langle s\rangle\rho' \quad \theta;\rho\models r:\langle s\rangle\rho'}{\theta;\rho\vdash\langle r :: a\rangle}$$

$$\theta;\rho\models\bullet:\langle\bullet\rangle\rho' \qquad \frac{\theta;\rho\models r:\langle s\rangle(\rho',x=e\rho')}{\theta;\rho\models r:\langle x=e,s\rangle\rho'} \qquad \frac{\theta;\rho\vdash e:a\rho' \quad \theta;\rho\models r:\langle s\rangle(\rho',x=e\rho)}{\theta;\rho\models(x=e,r):\langle x:a,s\rangle\rho'}$$

where the generic values $\mathrm{v}_k^{\cdot\cdot}$ do not occur in the respective rule's conclusion

....

$$\theta;\rho\vdash\perp\!\!\!\perp^v;b:a \qquad \theta;\rho\vdash(c\ e\ a)\rho';\bullet:a \qquad \frac{\theta;\rho\vdash(c'\ e'\ a')\rho';b:a}{\theta;\rho\vdash(c'\ e'\ a')\rho';(c\ x.e,b):a}(c\neq c')$$

$$\frac{\theta;(\rho,x=e'\rho')\vdash e:a\rho}{\theta;\rho\vdash(c\ e'\ a')\rho';(c\ x.e,b):a} \qquad \frac{w\!\uparrow\xrightarrow{\mu}\!\!\triangleright_\theta^*\{d\}\rho' \quad \forall c\in dom(d).\ \begin{pmatrix}\widehat{g}\longrightarrow\!\!\triangleright_{\theta_c}^* h \\ \theta_c;\rho\vdash h;b:a\end{pmatrix}}{\theta;\rho\vdash\widehat{g};b:a}$$

where $w = \textsc{scrutinee}(\widehat{g})$, $\theta_c = (\theta, w\approx(c\ x\ y)(x=\mathrm{v}_k^{\pi_c(\{d\}\rho')}, y=\{d\}\rho'))$ and $\mathrm{v}_k^{\cdot\cdot}$ does not occur in the above rule's conclusion

and $$\boxed{\begin{matrix}\textsc{scrutinee}(\gamma\ t\ b\ \widehat{g}\ \rho)=\textsc{scrutinee}(\widehat{g}) \\ \textsc{scrutinee}(w)=w\end{matrix}}$$

Figure 12: Syntactic type checking algorithm

therefore has to cover only three syntactic possibilities for h: it could be the value $\perp^t$, a constructor $(c\ e\ a)\rho'$, or a value in the grammar $\widehat{g}$.

If h is undefined, the rule succeeds immediately. For constructors, we traverse b to select the appropriate branch, which we type check by performing the appropriate binding in ρ. The most interesting rule covers values in the grammar $\widehat{g}$, where the technique used in our η-rule for verifying inclusions of cases is extended to type checking: we select the innermost scrutinee $w = \text{SCRUTINEE}(\widehat{g})$ of $\widehat{g}$, and proceed with the same rule after adding appropriate constraints to θ for each constructor in the type of w.

Our soundness result asserts that successful syntactic type checking of expression e in a well-typed environment ρ implies well-typing of the value $e\rho$.

Proposition 7 (Soundness of TC w.r.t. values). *Let ρ be a syntactic environment, e an expression, t a value, and δ a valuation. If $\vdash_\delta \theta$, $\vdash_\delta \rho$ and $\theta; \rho \vdash e$, then $\vdash_\delta e\rho$. If $\vdash_\delta \rho$, $\vdash_\delta t : \mathsf{V}$ and $\theta; \rho \vdash e : t$, then $\vdash_\delta e\rho : t\delta$.*

Finally, successful syntactic type checking implies correct typing in our semantic typing relation.

Theorem 8 (Soundness of TC w.r.t. model). *Let a and e be expressions and $\nu \in \mathsf{Env}$. If $(); () \vdash a : \star$ and $(); () \vdash e : a()$, then $e\nu : a\nu$.*

8 Related work

Apart from the sources cited in Section 1, which are referred to in detail throughout this paper, a number of other works that inspired our work are discussed here.

The language DML by Xi and Pfenning [30] enhances ML with a restricted form of dependent types. It is a conservative extension in the sense that it does not allow to type more programs than ML, but it permits more precise type annotations. This differs from our work, as we are aiming to enhance the flexibility of existing languages like Haskell and ML.

Recent work by McBride [22] develops a new high-level pattern matching notation for programming with dependent types within type theory. The logical consistency of type theory enables the programmer to view types as propositions and programs as proofs. This added power comes at the cost of abandoning the flexibility of general recursion, which may deter functional programmers from using such a system. That dependent types are a natural setting for generic programming is shown by Altenkirch and McBride [2] via an embedding of Generic Haskell [18] within McBride's framework.

Shao, Saha, Trifonov and Papaspyrou [29] propose a general framework for explicitly representing complex propositions and proofs in typed intermediate and assembly languages. Their framework allows proofs about programs involving computational effects and non-termination, but in order to maintain decidable type checking, their type language is never dependent on the computation language. This means that one needs to construct explicit mappings to mirror a term in the computation language by a term on the type level in order to allow type dependencies on that term. This method is highly suitable for representing complex propositions and proofs in typed intermediate languages, but appears somewhat heavy-duty for practical dependently-typed programming.

In the type theory community, Coquand, Pollack and Takeyama have proposed and investigated a logical framework with dependently-typed records [12] which includes additional notions such as subtyping of records not considered in this paper. Their goal is to preserve logical consistency in order to be able to reason about mathematical structures within their system, whereas we desire computational consistency of dependent records in the presence of general recursion.

A more efficient algorithm for subtyping recursive types than that of Amadio and Cardelli [3] has been proposed by Kozen, Palsberg and Schwartzbach [20]. It would be interesting to investigate whether their algorithm can be adapted to our setting.

McBride [21] demonstrates that Haskell is getting ever closer to a dependently-typed language, by exploiting functional dependencies [19] in Haskell's types classes to simulate a restricted form of dependent types. Hinze [17] uses the same technique to encode a variant of the formatting function *printf* in Haskell. These attempts show that the restricted machinery that is currently available results in somewhat convoluted programs, whereas the full power of dependent types offers a more natural and desirable solution.

Cheney and Hinze [10] propose to extend Haskell's type system with phantom types, which can be viewed as a special case of our dependent function types where only the type but not the *value* of a function depends on its input. Using a prototype implementation of our algorithm written in Haskell, we have translated and successfully type checked several of Cheney and Hinze's phantom type programs, as well as a well-typed interpreter by Augustsson and Carlsson [5], which requires the use of commuting conversions of cases.

9 Conclusions and future work

One of the main motivations of this work is to extend the expressiveness of current functional languages, by lifting restrictions of current type systems without sacrificing run-time type-safety in the sense of Milner [23]. The trade-off for such expressive power is that our typing relation, which is based on a natural interpretation of dependent types in domain theory, is undecidable.

Our algorithm is designed with the aim of type checking as many programs automatically as possible. To this end, we show how sophisticated techniques from work on the normalisation of simply-typed λ-terms with sums and subtyping of recursive types can be adapted to our setting, by modifying the former to ensure soundness under lazy evaluation and extending the latter to take into account functions occurring in types.

We found the main limitation of our algorithm to be the non-termination that arises when checking syntactic inclusions between functions that require explicit proof objects to establish their semantic validity. We plan to integrate type coercions into our language to allow programmers to aid the algorithm manually to that extent. Although sound proof objects cannot be written *within* our language because it is logically inconsistent in the sense that $\perp$ inhabits every type, an external theorem prover in the style of HOLCF [25] could be integrated with our algorithm to deal with domain theoretic proofs.

To show that an undecidable typing relation need not be a disadvantage in practice, we would like to construct an embedding of Haskell programs into our language using a source-to-source translation t with the following properties: if a program p is well-typed in Haskell then our algorithm type checks its translation $t(p)$ successfully, and t preserves the observational behaviour of programs. It would also be desirable to show that our algorithm terminates on at least all programs in the image of t. The formalisation by Faxén [14] of a static semantics for Haskell within a variant of the polymorphic λ-calculus provides a good starting point for such an investigation.

Acknowledgments

Thanks to Oege de Moor, Tom Melham and Ganesh Sittampalam for valuable feedback.

References

[1] T. Altenkirch, P. Dybjer, M. Hofmann, and P. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 303–310, 2001.

[2] T. Altenkirch and C. McBride. Generic programming within dependently typed programming. In *Proceedings of the IFIP TC2/WG2.1 Working Conference on Generic Programming, July 2002, Schloss Dagstuhl*, pages 1–20. Kluwer, 2003.

[3] R. M. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993.

[4] L. Augustsson. Cayenne—a language with dependent types. In *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming*, pages 239–250. ACM Press, 1998.

[5] L. Augustsson and M. Carlsson. An exercise in dependent types: A well-typed interpreter. In *Workshop on Dependent Types in Programming*, Gothenburg, 1999. Available from `http://www.cs.chalmers.se/~augustss/cayenne/`.

[6] L. Augustsson, T. Coquand, and B. Nordstrom. A short description of another logical framework. In *Informal Proceedings of the First Workshop on Logical Frameworks, Antibes*, 1990.

[7] V. Balat, R. Di Cosmo, and M. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 64–76. ACM Press, 2004.

[8] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of logic in computer science*, volume 2, pages 117–309. OUP, 1992.

[9] L. Cardelli. A polymorphic λ-calculus with type:type. Technical Report SRC-RR-10, Digital Equipment Corporation, SRC, May 1986.

[10] J. Cheney and R. Hinze. A lightweight implementation of generics and dynamics. In *Proceedings of the ACM SIGPLAN workshop on Haskell*, pages 90–104. ACM Press, 2002.

[11] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2-3):95–120, 1988.

[12] T. Coquand, R. Pollack, and M. Takeyama. A logical framework with dependently typed records. In *Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*, pages 105–119, 2003.

[13] T. Coquand and M. Takeyama. An implementation of type:type. In *Selected papers from the International Workshop on Types for Proofs and Programs*, pages 53–62. Springer-Verlag, 2002.

[14] K.-F. Faxen. A static semantics for Haskell. *Journal of Functional Programming*, 12(4&5):295–357, July 2002.

[15] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types*. Cambridge University Press, 1989.

[16] C. A. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, pages 633–674. Elsevier, 1990.

[17] R. Hinze. Formatting: a class act. *Journal of Functional Programming*, 13(5):935–944, 2003.

[18] R. Hinze and J. Jeuring. Generic Haskell: practice and theory. In *Generic Programming, Advanced Lectures*, volume 2793 of *LNCS*, pages 1–56. Springer Verlag, 2003.

[19] M. P. Jones. Type classes with functional dependencies. In G. Smolka, editor, *Proc. 9th European Symposium on Programming*, volume 1782 of *LNCS*, pages 230–244. Springer, March 2000.

[20] D. Kozen, J. Palsberg, and M. I. Schwartzbach. Efficient Recursive Subtyping. In *Proceedings of the 20th Symposium on Principles of Programming Languages*, pages 419–428. ACM Press, 1993.

[21] C. McBride. Faking it — Simulating dependent types in Haskell. *Journal of Functional Programming*, 12(4&5):375–392, July 2002.

[22] C. McBride and J. McKinna. The view from the left. *Journal of Functional Programming*, 14(1):69–111, January 2004.

[23] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[24] P. D. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, pages 575–631. Elsevier, 1990.

[25] O. Müller, T. Nipkow, D. v. Oheimb, and O. Slotosch. HOLCF = HOL + LCF. *Journal of Functional Programming*, 9(2):191–223, March 1999.

[26] S. Peyton-Jones. The Haskell 98 language and libraries: the revised report. *Journal of Functional Programming*, 13(1), January 2003.

[27] R. Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994.

[28] D. S. Scott. Lectures on a mathematical theory of computation. In M. Broy and G. Schmidt, editors, *Theoretical Foundations of Programming Methodology*, pages 145–292. D. Reidel, 1982.

[29] Z. Shao, B. Saha, V. Trifonov, and N. Papaspyrou. A type system for certified binaries. In *Proceedings of the 29th Symposium on Principles of Programming Languages*, pages 217–232. ACM Press, 2002.

[30] H. Xi and F. Pfenning. Dependent types in practical programming. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages*, pages 214–227. ACM Press, 1999.

A Auxiliary expressions

A.1 Syntax

$$e := \ldots \mid a * e \mid \pi_c e \mid \sigma_x\ e\ a \mid \Sigma_x\ e\ a$$

A.2 Semantics

$$\begin{array}{ll}
(a * e)\ \nu = a\nu * e\nu & (\sigma_x\ e\ a)\ \nu = \sigma_x\ e\nu\ a\nu \\
a * e \mid \nu = \mathsf{V} & \sigma_x\ e\ a \mid \nu = \Sigma_x\ e\nu\ a\nu \\
(\pi_c\ a)\ \nu = \pi_c\ a\nu & (\Sigma_x\ e\ a)\ \nu = \Sigma_x\ e\nu\ a\nu \\
\pi_c\ a \mid \nu = \mathsf{V} & \Sigma_x\ e\ a \mid \nu = \mathsf{V}
\end{array}$$

A.3 Semantic type checking

$$\frac{\nu \vdash a \quad a\nu = \mathsf{Prod}\ (e|\nu)\ f \quad \nu \vdash e}{\nu \vdash a * e} \qquad \frac{\nu \vdash a \quad a\nu = \mathsf{Data}\ \Delta \quad c\Delta = \mathsf{Def}\ u}{\nu \vdash \pi_c\ a}$$

$$\frac{\nu \vdash a \quad x \in xs \quad \nu \vdash e : a\nu = [xs : s]}{\nu \vdash \sigma_x\ e\ a} \qquad \frac{\nu \vdash a \quad x \in xs \quad \nu \vdash e : a\nu = [xs : s]}{\nu \vdash \Sigma_x\ e\ a}$$

A.4 Evaluation rules

$$\begin{array}{ll}
(a * e)\rho \longrightarrow a\rho * e\rho & (\sigma_x\ e\ a)\rho \longrightarrow \sigma_x\ (e\rho)\ (a\rho) \\
(\pi_c\ a)\rho \longrightarrow \pi_c(a\rho) & (\Sigma_x\ e\ a)\rho \longrightarrow \Sigma_x\ (e\rho)\ (a\rho) \\
a * e \mid \rho \longrightarrow \star & \sigma_x\ e\ a \mid \rho \longrightarrow \Sigma_x\ (e\rho)\ (a\rho) \\
\pi_c\ a \mid \rho \longrightarrow \star & \Sigma_x\ e\ a \mid \rho \longrightarrow \star
\end{array}$$

B Proofs

B.1 Auxiliary lemmas

Lemma 9. *We have* $g : \mathsf{Prod}\ u\ f$ *if and only if (1)* $g \in [\mathsf{D} \to \mathsf{D}]$*, (2)* $g = g \circ \mathsf{p}_u$ *and (3)* $\forall d \in \mathsf{D}.\ g\ d : f\ (\mathsf{p}_u d)$*.*

Proof. First suppose that $g : \mathsf{Prod}\ u\ f$. Since $\mathsf{p}_{\mathsf{Prod}\ u\ f} g = g$, we must have (1) $g \in [\mathsf{D} \to \mathsf{D}]$, for $\mathsf{p}_{\mathsf{Prod}\ u\ f} g = \bot$ if $g \notin [\mathsf{D} \to \mathsf{D}]$. By (1) and definition of $\mathsf{p}_{\mathsf{Prod}\ u\ f} g$, we have (4) $\forall d \in \mathsf{D}.\ g\ d = \mathsf{p}_{f\ (\mathsf{p}_u d)} g\ (\mathsf{p}_u d)$. In particular, (4) holds for all $d : u$, giving (5) $\forall d \in \mathsf{D}.\ g\ (\mathsf{p}_u d) : f\ (\mathsf{p}_u d)$. Substituting (5) into (4) gives (2), and this together with (5) gives (3).

Conversely, suppose conditions (1–3) hold. Then

$$
\begin{aligned}
& \mathsf{p}_{\mathsf{Prod}\ u\ f}\, g \\
= \quad & \hspace{20em} \{\text{def } \mathsf{p}_{(-)} \text{ and } (1)\} \\
& \lambda d. \mathsf{p}_{(f\ (\mathsf{p}_u\, d))}(g\ (\mathsf{p}_u\, d)) \\
= \quad & \hspace{20em} \{(2)\} \\
& \lambda d. \mathsf{p}_{(f\ (\mathsf{p}_u\, d))}(g\ d) \\
= \quad & \hspace{20em} \{(3)\} \\
& \lambda d. (g\ d) \\
= \quad & \hspace{20em} \{(1)\} \\
& g
\end{aligned}
$$

as required. □

Lemma 10. *Let $g_i \in \mathsf{D}$, $u_i \in T$ and $f_i \in [\mathsf{D} \to \mathsf{T}]$, and suppose that $u_1 \sqsubseteq u_2$, $f_1 \sqsubseteq f_2$ and $g_i : \mathsf{Prod}\ u_i\ f_i$, where $i \in \{1, 2\}$. Then*

$$
\begin{aligned}
g_1 \sqsubseteq g_2 &\iff \forall d.\ g_1\ (\mathsf{p}_{u_1}\, d) \sqsubseteq g_2\ (\mathsf{p}_{u_1}\, d) \\
&\iff \forall d.\ g_1\ (\mathsf{p}_{u_1}\, d) \sqsubseteq g_2\ (\mathsf{p}_{u_2}\, d).
\end{aligned}
$$

Proof. The first ($\Longrightarrow$) direction holds trivially, and the second one holds by assumption $u_1 \sqsubseteq u_2$ and monotonicity of function application and $\mathsf{p}_{(-)}$. Now let $d \in \mathsf{D}$ and suppose $g_1\ (\mathsf{p}_{u_1}\, d) \sqsubseteq g_2\ (\mathsf{p}_{u_2}\, d)$. Then we have

$$
\begin{aligned}
& g_1\ d \\
= \quad & \hspace{20em} \{g_1 : \mathsf{Prod}\ u_1\ f_1\} \\
& \mathsf{p}_{(f_1\ (\mathsf{p}_{u_1}\, d))}(g_1\ (\mathsf{p}_{u_1}\, d)) \\
\sqsubseteq \quad & \hspace{20em} \{\text{assumption}\} \\
& \mathsf{p}_{(f_1\ (\mathsf{p}_{u_1}\, d))}(g_2\ (\mathsf{p}_{u_2}\, d)) \\
\sqsubseteq \quad & \quad \{u_1 \sqsubseteq u_2,\ f_1 \sqsubseteq f_2 \text{ and monotonicity of function application and } \mathsf{p}_{(-)}\} \\
& \mathsf{p}_{(f_2\ (\mathsf{p}_{u_2}\, d))}(g_2\ (\mathsf{p}_{u_2}\, d)) \\
= \quad & \hspace{20em} \{g_2 : \mathsf{Prod}\ u_2\ f_2\} \\
& g_2\ d
\end{aligned}
$$

and thus $g_1 \sqsubseteq g_2$ as required. □

Lemma 11. *For $d_1, d_2 \in \mathsf{D}$ and $u \in \mathsf{T}$, we have that $d_1 : u$ implies $d_1 \cdot d_2 : u * d_2$.*

Proof. By cases on u.

- Cases $\bot$, V, $\mathsf{Data}\ \Delta$, $[xs : s]$. Since $d_1 : u$, we must have $d_1 = \bot$ or $d_1 \notin [\mathsf{D} \to \mathsf{D}]$. In both cases, we have $d_1 \cdot d_2 = (\mathsf{p}_{[\mathsf{D} \to \mathsf{D}]}\, d_1)\ d_2 = \bot : u * d_2$, as desired.

- Case $\mathsf{Prod}\ u'\ f$. We have

$$
\begin{array}{ll}
& \mathsf{p}_{(\mathsf{Prod}\ u'\ f)*d_2}(d_1 \cdot d_2) \\
= & \hfill \{d_1 : \mathsf{Prod}\ u'\ f\} \\
& \mathsf{p}_{f\,(\mathsf{p}_{u'}\,d_2)}(\mathsf{p}_{f\,(\mathsf{p}_{u'}\,d_2)}(d_1\ (\mathsf{p}_{u'}\,d_2))) \\
= & \hfill \{\text{idempotency of } \mathsf{p}_{(-)}\} \\
& \mathsf{p}_{f\,(\mathsf{p}_{u'}\,d_2)}(d_1\ (\mathsf{p}_{u'}\,d_2)) \\
= & \hfill \{d_1 : \mathsf{Prod}\ u'\ f\} \\
& d_1 \cdot d_2
\end{array}
$$

as required.

□

Lemma 12. *For $x \in Ide$, $d \in \mathsf{D}$, $\varrho \in \mathsf{D}^{Ide}$, $xs \in (Ide^{\pm})^*$, $s \in \mathsf{Sig}_{xs}\ (sig\ xs)$ and $x \notin xs$, we have $\mathsf{p}_{[xs:s]}(\varrho, x = d) = \mathsf{p}_{[xs:s]}\varrho$.*

Proof. Straightforward induction on xs. □

Lemma 13. *For $d \in \mathsf{D}$ and $u \in \mathsf{T}$, we have $\tau\ u\ d : u$.*

Proof. Suppose $u = [xs : s]$ and $d = \varrho \in \mathsf{D}^{Ide}$, for otherwise $\tau\ u\ d = \bot$ and the result holds vacuously. We prove $\tau\ [xs : s]\ \varrho : [xs : s]$ by induction on xs.

- Case $\langle\rangle$. Immediate.
- Case $\langle x^-, xs\rangle$. We have

$$
\begin{array}{ll}
& \mathsf{p}_{[x:u,xs:f]}(\tau\ [x : u, xs : f]\ \varrho) \\
= & \hfill \{\text{def } \tau\} \\
& \mathsf{p}_{[x:u,xs:f]}(\tau\ [xs : f\ (\mathsf{p}_u x \varrho)]\ \varrho, x = \mathsf{p}_u x \varrho) \\
= & \hfill \{\text{def } \mathsf{p}_{[x:u,xs:f]}\} \\
& (\mathsf{p}_{[xs:f\ (\mathsf{p}_u(\mathsf{p}_u x \varrho))]}(\tau\ [xs : f\ (\mathsf{p}_u x \varrho)]\ \varrho, x = \mathsf{p}_u x \varrho), x = \mathsf{p}_u x \varrho) \\
= & \hfill \{\text{Lemma 12}\} \\
& (\mathsf{p}_{[xs:f\ (\mathsf{p}_u x \varrho)]}(\tau\ [xs : f\ (\mathsf{p}_u x \varrho)]\ \varrho), x = \mathsf{p}_u x \varrho) \\
= & \hfill \{\text{induction hypothesis on } xs\} \\
& (\tau\ [xs : f\ (\mathsf{p}_u x \varrho)]\ \varrho, x = \mathsf{p}_u x \varrho) \\
= & \hfill \{\text{def } \tau\} \\
& \tau\ [x : u, xs : f]\ \varrho
\end{array}
$$

  as required.
- Case $\langle x^+, xs\rangle$. Similar to previous case, using continuity of $\sqcap$.

□

Lemma 14. *For $x \in Ide$, $d \in \mathsf{D}$ and $u \in \mathsf{T}$, we have $\sigma_x\ d\ u : \Sigma_x\ d\ u$.*

Proof. Suppose $u = [xs : s]$ and $d = \varrho \in \mathsf{D}^{Ide}$, for otherwise $\sigma_x\ d\ u = \bot$ and the result holds vacuously. We prove $\sigma_x\ \varrho\ [xs : s] : \Sigma_x\ \varrho\ [xs : s]$ by induction on xs.

- Case $\langle\rangle$. Immediate.
- Case $\langle y^-, xs\rangle$. If $x \neq y$ then the result follows from induction hypothesis on xs, and if $x = y$ it holds immediately.
- Case $\langle y^+, xs\rangle$. Similar to the previous case.

□

B.2 Proof of Lemma 1

For all expressions e and $\nu \in \mathsf{Env}$, we have $e\nu : e \mid \nu$.

Proof. By induction on e.

- Cases x, $\star$, $\forall x : a.b$, $e_1 * e_2$, $\langle s\rangle$, $c\ e\ a$, $\{d\}$, $case\ a\ b\ e$, $\pi_c\ e$, $\Sigma_x\ e\ a$. Immediate.
- Case $\lambda x : a.e$. We have to establish $g : \mathsf{Prod}\ u\ f$, for $u = \mathsf{p}_{\mathsf{V}} a\nu$, $g = \lambda d.e(\nu, x : u = d)$ and $f = \lambda d.e \mid (\nu, x : u = d)$. By Lemma 9, it suffices to show (1) $g \in [\mathsf{D} \to \mathsf{D}]$, (2) $g = g \circ \mathsf{p}_u$ and (3) $\forall d \in \mathsf{D}.\ g\ d : f\ (\mathsf{p}_u d)$. Condition (1) holds by definition, (2) $\forall d \in \mathsf{D}.\ e(\nu, x : u = d) = e(\nu, x : u = \mathsf{p}_u d)$ follows from definition of environment update, and (3) $\forall d \in \mathsf{D}.\ e(\nu, x : u = d) : e \mid (\nu, x : u = \mathsf{p}_u d)$ follows from induction hypothesis on e.
- Case $e_1\ e_2$. Follows from induction hypothesis on e_1 and Lemma 11.

- Case $\mu x : a.e$. Let $g = \lambda d.\mathsf{p}_{\mathsf{p_V}\, a\nu}\, e(\nu, x : \mathsf{p_V}\, a\nu = d)$. Then

$$
\begin{aligned}
& \mathsf{p}_{(\mu x : a.e \mid \nu)}(\mu x : a.e\nu) \\
= & \qquad \{\text{semantics of } \mu\} \\
& \mathsf{p}_{\mathsf{p_V}\, a\nu}(\mathsf{fix}\, g) \\
= & \qquad \{\text{property of } \mathsf{fix}\} \\
& \mathsf{p}_{\mathsf{p_V}\, a\nu}(g(\mathsf{fix}\, g)) \\
= & \qquad \{\text{def } g\} \\
& \mathsf{p}_{\mathsf{p_V}\, a\nu}(\mathsf{p}_{\mathsf{p_V}\, a\nu}\, e(\nu, x : \mathsf{p_V}\, a\nu = \mathsf{fix}\, g)) \\
= & \qquad \{\text{idempotency of } \mathsf{p}_{(-)}\} \\
& \mathsf{p}_{\mathsf{p_V}\, a\nu}\, e(\nu, x : \mathsf{p_V}\, a\nu = \mathsf{fix}\, g) \\
= & \qquad \{\text{def } g\} \\
& g(\mathsf{fix}\, g) \\
= & \qquad \{\text{property of } \mathsf{fix}\} \\
& \mathsf{fix}\, g \\
= & \qquad \{\text{semantics of } \mu\} \\
& \mu x : a.e\nu
\end{aligned}
$$

  as required.

- Case $\langle r :: a \rangle$. By Lemma 13.
- Cases $e.x$, $\sigma_x\ e\ a$. By Lemma 14.

□

B.3 Proof of Lemma 2

For all values v and $\delta \in \mathsf{Val}$, $v\delta : v \mid \delta$.

Proof. By induction on v.

- Cases v_k^v, $\star$, $e \mid \rho$, $v_1 * v_2$, $v\!\uparrow$, $\perp\!\!\!\perp^v$, $\gamma\ t\ b\ v\ \rho$, $\pi_c\ v$, $\Sigma_x\ v_1\ v_2$, $\Sigma_x^{zs}\ v\ s\ \rho$. Immediate.
- Case $e\rho$. By Lemma 1.
- Case $v_1\ v_2$. By induction hypothesis on v_1 and Lemma 11.
- Case $\sigma_x\ v_1\ v_2$, $\sigma_x\ r\ \rho\ s\ \rho'$. By Lemma 14.

□

B.4 Further auxiliary lemmas

Lemma 15. *For $\varrho \in \mathsf{D}^{Ide}$, $d \in \mathsf{D}$, $x \in Ide$, $xs \in (Ide^{\pm})^*$, $s \in \mathsf{Sig}_{xs}$ $(sig\ xs)$ and $x \notin xs$, we have $\tau\ [xs : s]\ (\varrho, x = d) = \tau\ [xs : s]\ \varrho$.*

Proof. Straightforward induction on xs. □

Lemma 16. *For $\varrho \in \mathsf{D}^{Ide}$, $d \in \mathsf{D}$, $x, y \in Ide$, $xs \in (Ide^{\pm})^*$, $s \in \mathsf{Sig}_{xs}$ $(sig\ xs)$ and $x \notin xs$, we have $\sigma_y\ (\varrho, x = d)\ [xs : s] = \sigma_y\ \varrho\ [xs : s]$ and $\Sigma_y\ (\varrho, x = d)\ [xs : s] = \Sigma_y\ \varrho\ [xs : s]$.*

Proof. We prove $\sigma_y\ (\varrho, x = d)\ [xs : s] = \sigma_y\ \varrho\ [xs : s]$ by induction on xs. The proof for $\Sigma_y\ (\varrho, x = d)\ [xs : s] = \Sigma_y\ \varrho\ [xs : s]$ is similar.

- Case $\langle\rangle$. Immediate.
- Case $\langle z^-, zs\rangle$, $y = z$. Then

$$\begin{array}{ll} & \sigma_y\ (\varrho, x = d)\ [z : u, zs : f] \\ = & \qquad\qquad \{\text{def } \sigma_y\ ,\ y = z\} \\ & \mathsf{p}_u(y(\varrho, x = d)) \\ = & \qquad\qquad \{y = z,\ x \notin xs = \langle z^-, zs\rangle\} \\ & \mathsf{p}_u(y\varrho) \\ = & \qquad\qquad \{\text{def } \sigma_y\ ,\ y = z\} \\ & \sigma_y\ \varrho\ [z : u, zs : f] \end{array}$$

- Case $\langle z^-, zs\rangle$, $y \neq z$. Then

$$\begin{array}{ll} & \sigma_y\ (\varrho, x = d)\ [z : u, zs : f] \\ = & \qquad\qquad \{\text{def } \sigma_y\ ,\ y \neq z\} \\ & \sigma_y\ (\varrho, x = d)\ [zs : f\ (\mathsf{p}_u(z(\varrho, x = d)))] \\ = & \qquad\qquad \{\text{induction hypothesis on } zs,\ x \notin xs = \langle z^-, zs\rangle\} \\ & \sigma_y\ \varrho\ [zs : f\ (\mathsf{p}_u(z\varrho))] \\ = & \qquad\qquad \{\text{def } \sigma_y\ ,\ y \neq z\} \\ & \sigma_y\ \varrho\ [z : u, zs : f] \end{array}$$

- Case $\langle z^+, zs\rangle$. Similar to previous two cases.

□

Lemma 17. *Let $c \notin dom(b)$. Then we have $\Delta; \nu \vdash d; b : u$ if and only if $(\forall d'.\ d \neq \mathsf{Inj}_c\ d' \implies \Delta; \nu \vdash d; b : u) \wedge (\forall c' \in dom(b).\ c'\Delta = \mathsf{Def}\ u')$.*

Proof. By induction on b.

- Case $(\bullet)$. Immediate.
- Case $(c'\ x.e, b)$.
  - Subcase $c = c'$. Vacuously true, as it contradicts the assumption $c \notin dom(c'\ x.e, b)$.
  - Subcase $c \neq c'$. By inversion of derivations and induction hypothesis.

□

B.5 Proof of Proposition 3

Let v, v' be values. Then for all valuations δ,

$$\vdash_\delta v \ \wedge\ v \xrightarrow{\mu}\!\!\triangleright v' \implies \vdash_\delta v' \ \wedge\ v\delta = v'\delta \ \wedge\ v \mid \delta = v' \mid \delta.$$

Proof. By induction on v. We use the following trivial consequences of Lemma 1 and Lemma 2: $\nu \vdash e : u$ implies $e\nu : u$ and $\vdash_\delta v : u$ implies $v\delta : u$.

- Cases $x \mid (\rho, x = v_1)$, $\star\rho$, $\star \mid \rho$, $\forall x : a_1.a_2 \mid \rho$, $(e_1\ \ e_2)\rho$, $e_1\ \ e_2 \mid \rho$, $\{d\} \mid \rho$, $(case\ \ a\ \ b\ \ e)\rho$, $\langle s\rangle \mid \rho$, $e.x\ \rho$, $e.x \mid \rho$, $(a * e)\rho$, $(\pi_c\ \ a)\rho$, $(\sigma_x\ \ e\ \ a)\rho$, $(\Sigma_x\ \ e\ \ a)\rho$, $(a * e) \mid \rho$, $(\pi_c\ a) \mid \rho$, $(\sigma_x\ e\ a) \mid \rho$, $(\Sigma_x\ e\ a) \mid \rho$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ hold by definition.

- Cases $c\ \ e\ \ a \mid \rho$, $(case\ \ a\ \ b\ \ e) \mid \rho$, $\langle r :: a\rangle \mid \rho$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$ and (1) $\rho\delta \vdash a : \mathsf{V}$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from (1).

- Case $x(\rho, x = v_1)$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$. The equality $v \mid \delta = v' \mid \delta$ holds by definition and $v\delta = v'\delta$ follows from Lemma 2.

- Cases $x(\rho, y = v_1)$, $x \mid (\rho, y = v_1)$, where $y \neq x$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from the assumption $y \neq x$.

- Case $v_1\ v_2$.
  - Subcase v_1 is $\perp\!\!\!\perp^t$. For $\vdash_\delta v'$, we need to show (1) $\vdash_\delta t$, (2) $\vdash_\delta v_2$ and (3) $t\delta = \mathsf{Prod}\ (v_2 \mid \delta)\ f$. Inversion of $\vdash_\delta v$ gives (1) and (2), as well as (4) $\vdash_\delta t : \mathsf{V}$ and (5) $\mathsf{p}_{\mathsf{V}} t\delta = \mathsf{Prod}\ (v_2 \mid \delta)\ f$. Conditions (4) and (5) together imply (3). The equation $v\delta = v'\delta$ holds by definition, and $v \mid \delta = v' \mid \delta$ follows from (4).
  - Subcase v_1 is $\lambda x : a.e\rho$. Then v' must be $e(\rho, x = v_2)$. For $\vdash_\delta v'$, we need to show (1) $\vdash_\delta \rho$, (2) $\vdash_\delta v_2$ and (3) $(\rho\delta, x : v_2 \mid \delta = v_2\delta) \vdash e$. Inversion of $\vdash_\delta v$ gives (1) and (2), as well as (4) $\forall d.\ (\rho\delta, x : \mathsf{p}_{\mathsf{V}} a\rho\delta = d) \vdash e$ and (5) $\lambda x : a.e\rho \mid \delta = \mathsf{Prod}\ (v_2 \mid \delta)\ f$. Since Prod is injective, (5) implies that (6) $\mathsf{p}_{\mathsf{V}} a\rho\delta = v_2 \mid \delta$. Condition (3) now follows from (4) and (6). The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from (6).
  - Subcase $(\gamma\ t\ b\ w\ \rho)\ v_2$. Covered separately below (due to size of proof).

- Subcase $v_1 \longrightarrow\!\triangleright v_1'$. Then v' must be $v_1'\ v_2$. The desired conclusions follow from inversion of $\vdash_\delta v$ and induction hypothesis on v_1.

- Case $t * v_2$. Similar to the previous case.
- Case $\mu x : a.e\rho$. Then v' must be $e(\rho, x = \mu x : a.e\rho)$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$, as well as (1) $\rho\delta \vdash a : \mathsf{V}$ and (2) $\forall d.\ (\rho\delta, x : \mathsf{p_V}\, a\rho\delta = d) \vdash e : a\rho\delta$. The equation $v\delta = v'\delta$ follows from the property $\forall g \in [\mathsf{D} \to \mathsf{D}].\ \mathsf{fix}\, g = g(\mathsf{fix}\, g)$, and $v \mid \delta = v' \mid \delta$ follows from (1) and (2).
- Case $\mu x : a.e \mid \rho$. Similar to the previous case.
- Case $\gamma\ t\ b_0\ v_0\ \rho$.
  - Subcases $\gamma\ t\ \bullet\ v_0\ \rho$, $\gamma\ t\ (c\ x.e, b)\ \perp\!\!\!\perp^{v_1}\ \rho$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ hold by definition.
  - Subcase $\gamma\ t\ (c\ x.e, b)\ ((c\ e'\ a)\rho')\ \rho$. For $\vdash_\delta v$, we need to establish (1) $\vdash_\delta (\rho, x = e'\rho')$ and (2) $(\rho, x = e'\rho')\delta \vdash e$. Inversion of $\vdash_\delta v$ gives (1), as well as (3) $\vdash_\delta t : \mathsf{V}$, (4) $\vdash_\delta ((c\ e'\ a)\rho') : \mathsf{Data}\ \Delta$ and (5) $\Delta; \rho\delta \vdash ((c\ e'\ a)\rho'); (c\ x.e, b) : t\delta$. Inversion of (4) further gives (6) $\rho'\delta \vdash a : \mathsf{V}$, (7) $a\rho'\delta = \mathsf{Data}\ \Delta$, (8) $c\Delta = \mathsf{Def}\ u$ and (9) $\rho'\delta \vdash e' : u$. We also have (10) $(c\ e'\ a)\rho'\delta = \mathsf{Inj}_c\ (e'\rho'\delta)$ since

$$
\begin{aligned}
& (c\ e'\ a)\rho'\delta \\
= & \qquad \{\text{semantics of expressions}\} \\
& \mathsf{p}_{\mathsf{p_V}\, a\rho'\delta}(\mathsf{Inj}_c\ (e'\rho'\delta)) \\
= & \qquad \{(6),\ (7)\} \\
& \mathsf{p}_{\mathsf{Data}\ \Delta}(\mathsf{Inj}_c\ (e'\rho'\delta)) \\
= & \qquad \{(8),\ (9)\} \\
& (\mathsf{Inj}_c\ (e'\rho'\delta))
\end{aligned}
$$

  Conditions (5), (9) and (10) together imply (2) as well as $v \mid \delta = v' \mid \delta$. For $v\delta = v'\delta$ we have

$$
\begin{aligned}
& (\gamma\ t\ (c\ x.e, b)\ ((c\ e'\ a)\rho')\ \rho)\delta \\
= & \qquad \{\text{semantics of values}\} \\
& \mathsf{p}_{\mathsf{p_V}\, t\delta}([\![c\ x.e, b]\!]\ ((c\ e'\ a)\rho'\delta)\ (c\ e'\ a \mid \rho'\delta)\ \rho\delta) \\
= & \qquad \{(3),\ (4),\ (10)\} \\
& \mathsf{p}_{t\delta}([\![c\ x.e, b]\!]\ (\mathsf{Inj}_c\ (e'\rho'\delta))\ (\mathsf{Data}\ \Delta)\ \rho\delta) \\
= & \qquad \{\text{def } [\![c\ x.e, b]\!]\ (-)\ (-)\ (-)\} \\
& \mathsf{p}_{t\delta}\, e(\rho\delta, x : u = e'\rho'\delta) \\
= & \qquad \{(9)\} \\
& \mathsf{p}_{t\delta}\, e(\rho, x = e'\rho')\delta \\
= & \qquad \{(5),\ (9),\ (10)\} \\
& e(\rho, x = e'\rho')\delta
\end{aligned}
$$

as required.

– Subcase $\gamma\ t\ (c\ x.e, b)\ ((c'\ e'\ a)\rho')\ \rho$ with $c \neq c'$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$. The equality $v\delta = v'\delta$ holds since

$$\begin{array}{ll} & (\gamma\ t\ (c\ x.e, b)\ ((c'\ e'\ a)\rho')\ \rho)\delta \\ = & \hfill \{\text{semantics of values}\} \\ & \mathsf{p}_{\mathsf{p_V}t\delta}(\llbracket c\ x.e, b\rrbracket\ ((c'\ e'\ a)\rho'\delta)\ (c'\ e'\ a \mid \rho'\delta)\ \rho\delta) \\ = & \hfill \{\text{by similar reasoning as in previous case}\} \\ & \mathsf{p}_{t\delta}(\llbracket c\ x.e, b\rrbracket\ (\mathsf{Inj}_{c'}\ (e'\rho'\delta))\ (\mathsf{Data}\ \Delta)\ \rho\delta) \\ = & \hfill \{c \neq c'\} \\ & \mathsf{p}_{t\delta}(\llbracket b\rrbracket\ (\mathsf{Inj}_{c'}\ (e'\rho'\delta))\ (\mathsf{Data}\ \Delta)\ \rho\delta) \\ = & \hfill \{\text{by similar reasoning as in previous case}\} \\ & (\gamma\ t\ b\ ((c'\ e'\ a)\rho')\ \rho)\delta \end{array}$$

and $v \mid \delta = v' \mid \delta$ holds by definition.

– Subcase $v_0 \longrightarrow\!\!\!\triangleright\, v_0'$. The desired conclusions follow from inversion of $\vdash_\delta v$ and induction hypothesis on v_0.

- Case $\pi_c\ t$.

  – Subcase $\pi_c\ (\{c\ \ a, d\})$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$ and (1) $\rho\delta \vdash a : \mathsf{V}$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from (1).

  – Subcase $\pi_c\ (\{c'\ a, d\})$ with $c \neq c'$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$. The equality $v\delta = v'\delta$ follows from assumption $c \neq c'$, and $v \mid \delta = v' \mid \delta$ holds by definition.

  – Subcase $\pi_c\ (\gamma\ t\ b\ w\ \rho)$. Covered separately below.

  – Subcase $t \xrightarrow{\mu}\!\triangleright\, t'$. The desired conclusions follow from inversion of $\vdash_\delta v$ and induction hypothesis on t.

- Case $\sigma_x\ v_1\ v_2$.

  – Subcase $\sigma_x\ (\langle r :: a\rangle\rho)\ \langle s\rangle\rho'$. Then v' must be $\sigma_x\ r\ \rho\ s\ \rho'$. For $\vdash_\delta v'$, we need to establish (1) $\vdash_\delta \rho$, (2) $\vdash_\delta \langle s\rangle\rho'$, (3) $\rho\delta \models r : \langle s\rangle\rho'\delta$ and (4) $x \in \mathit{dom}(s)$. Inversion of $\vdash_\delta v$ immediately gives (1), (2) and (4), as well as $\rho\delta \models r : a\rho\delta$, $\rho\delta \vdash a : \mathsf{V}$ and (5) $\mathsf{p_V} a\rho\delta = \langle s\rangle\rho'\delta$. The latter three conditions together give (3). The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from (5).

  – Subcase $\sigma_x\ \perp\!\!\!\perp^t\ v_2$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ hold because, for all signatures $[xs : s]$, we have $\sigma_x \perp [xs : s] = \perp$ by an easy induction on xs.

  – Subcases $\sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v_2$, $\sigma_x\ v_1\ (\gamma\ t\ b\ w\ \rho)$. Covered separately below.

  – Subcase $v_1 \longrightarrow\!\!\!\triangleright\, v_1'$. The desired conclusions follow from inversion of $\vdash_\delta v$ and induction hypothesis on v_1.

- Subcase $v_1 \not\longrightarrow\!\!\triangleright\, v_1'$ and $v_2 \longrightarrow\!\!\triangleright\, v_2'$. The desired conclusions follow from inversion of $\vdash_\delta v$ and induction hypothesis on v_2.

- Case $\Sigma_x\ v_1\ v_2$.
  - Subcase v_2 is $\langle s\rangle\rho$. Inversion of $\vdash_\delta v$ gives $\vdash_\delta v'$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ hold by definition.
  - Subcase $\Sigma_x\ v_1\ (\gamma\ t\ b\ w\ \rho)$. Covered separately below.
  - Subcase $v_2 \xrightarrow{\mu}\!\!\triangleright\, v_2'$. The desired conclusions follow from inversion of $\vdash_\delta v$ and induction hypothesis on v_2.

- Case $\sigma_{x_0}\ r_0\ \rho\ s_0\ \rho'$.
  - Subcase $\sigma_x\ r\ \rho\ (x = e, s)\ \rho'$. Then v' must be $e\rho'$. Inversion of $\vdash_\delta v$ gives $\vdash_\delta v'$. The equality $v\delta = v'\delta$ follows from Lemma 1 and $v \mid \delta = v' \mid \delta$ holds by definitions.
  - Subcase $\sigma_x\ (x = e, r)\ \rho\ (x : a, s)\ \rho'$. Then v' must be $e\rho$. Inversion of $\vdash_\delta v$ gives $\vdash_\delta v'$, as well as (1) $e \mid \rho\delta = \mathsf{p}_{\mathsf{V}} a\rho'\delta$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from (1) and Lemma 1.
  - Subcase $\sigma_y\ r\ \rho\ (x = e, s)\ \rho'$ with $y \neq x$. Then v' must be $\sigma_y\ r\ \rho\ s\ (\rho', x = e\rho')$. Inversion of $\vdash_\delta v$ gives $\vdash_\delta v'$. The equality $v\delta = v'\delta$ follows from Lemma 1 and $v \mid \delta = v' \mid \delta$ holds by definitions.
  - Subcase $\sigma_y\ (x = e, r)\ \rho\ (x : a, s)\ \rho'$ with $y \neq x$. Then v' must be $\sigma_y\ r\ \rho\ s\ (\rho', x = e\rho)$. For $\vdash_\delta v'$, we need to show (1) $\vdash_\delta \rho$, (2) $\vdash_\delta \langle s\rangle(\rho', x = e\rho)$, (3) $\rho\delta \models r : \langle s\rangle(\rho', x = e\rho)\delta$ and (4) $y \in dom(s)$. Inversion of $\vdash_\delta v$ gives conditions (1) and (4), as well as (5) $\forall d.\ (\rho'\delta, x : \mathsf{p}_{\mathsf{V}} a\rho'\delta = d) \vdash \langle s\rangle$, (6) $\rho\delta \vdash e : \mathsf{p}_{\mathsf{V}} a\rho'\delta$ and (7) $\rho\delta \models r : \langle s\rangle(\rho'\delta, x : \mathsf{p}_{\mathsf{V}} a\rho'\delta = e\rho\delta)$. Combining conditions (5) and (6) imply (2), and (6) and (7) together imply (3). The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from condition (6), Lemmas 1, 15 and 16, and the fact that we require distinct field identifiers in signatures.

- Case $\Sigma_{x_0}^{zs}\ v\ s_0\ \rho$.
  - Subcase $\Sigma_x^{zs}\ v_1\ (x = e, s)\ \rho$. Then v' must be $e \mid \rho$. Inversion of $\vdash_\delta v$ gives $\vdash_\delta v'$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ hold by definition.
  - Subcase $\Sigma_x^{zs}\ v_1\ (x : a, s)\ \rho$. Then v' must be $a\rho$. Inversion of $\vdash_\delta v$ gives $\vdash_\delta v'$ and (1) $\rho\delta \vdash a : \mathsf{V}$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from (1).
  - Subcase $\Sigma_y^{zs}\ v_1\ (x = e, s)\ \rho$ with $y \neq x$. Then v' must be $\Sigma_y^{zs,x}\ v_1\ s\ (\rho, x = e\rho)$. For $\vdash_\delta v'$, we need to show (1) $\vdash_\delta v_1 : [ys : \_]$, (2) $\vdash_\delta \langle s\rangle(\rho, x = e\rho)$, (3) $y \in xs$ and $\langle zs, x\rangle +\!\!+\ xs = ys$ (where $\langle s\rangle(\rho, x = e\rho)\delta = [xs : \_]$) and (4a) $\forall z \notin \langle zs, x\rangle.\ \Sigma_z\ (v_1\delta)\ (v_1 \mid \delta) = \Sigma_z\ (v_1\delta)\ (\langle s\rangle(\rho, x = e\rho)\delta)$ and (4b) $\forall z \notin \langle zs, x\rangle.\ \sigma_z\ (v_1\delta)\ (v_1 \mid \delta) = \sigma_z\ (v_1\delta)\ (\langle s\rangle(\rho, x = e\rho)\delta)$. Conditions (1–3) follow from assumption $y \neq x$ and inversion of $\vdash_\delta v$, which also gives (5a) $\forall z \notin \langle zs, x\rangle.\ \Sigma_z\ (v_1\delta)\ (v_1 \mid \delta) = \Sigma_z\ (v_1\delta)\ (\langle x = e, s\rangle\rho\delta)$ and (5b)

$\forall z \notin \langle zs, x\rangle.\ \sigma_z\ (v_1\delta)\ (v_1 \mid \delta) = \sigma_z\ (v_1\delta)\ (\langle x = e, s\rangle\rho\delta)$. To establish (4a), consider $z \notin \langle zs, x\rangle$, so that $z \neq x$. Then we have

$$
\begin{array}{ll}
& \Sigma_z\ (v_1\delta)\ (v_1 \mid \delta) \\
= & \hfill \{(5a)\} \\
& \Sigma_z\ (v_1\delta)\ (\langle x = e, s\rangle\rho\delta) \\
= & \hfill \{\text{semantics of expressions}\} \\
& \Sigma_z\ (v_1\delta)\ [x : (e \mid \rho\delta) = e\rho\delta, \langle s\rangle(\rho\delta, x : (e \mid \rho\delta) = e\rho\delta)] \\
= & \hfill \{z \neq x\} \\
& \Sigma_z\ (v_1\delta)\ (\langle s\rangle(\rho\delta, x : (e \mid \rho\delta) = e\rho\delta)) \\
= & \hfill \{\text{semantics of environments}\} \\
& \Sigma_z\ (v_1\delta)\ (\langle s\rangle(\rho, x = e\rho)\delta)
\end{array}
$$

as required. The proof for (4b) is similar. By similar reasoning (but using the assumption $y \neq x$ instead of $z \neq x$) we also obtain the required equality $v\delta = v'\delta$. By definition we have $v \mid \delta = v' \mid \delta$.

– Subcase $\Sigma_y^{zs}\ v_1\ (x : a, s)\ \rho$ with $y \neq x$. Then v' must be $\Sigma_y^{zs,x}\ v_1\ s\ (\rho, x = \sigma_x\ v_1\ (v_1\uparrow))$. For $\vdash_\delta v'$, we need to show (1) $\vdash_\delta v_1 : [ys : \_]$, (2) $\vdash_\delta \langle s\rangle(\rho, x = \sigma_x\ v_1\ (v_1\uparrow))$, (3) $y \in xs$ and $\langle zs, x\rangle +\!\!+\, xs = ys$ (where $\langle s\rangle(\rho, x = \sigma_x\ v_1\ (v_1\uparrow))\delta = [xs : \_]$) and (4a) $\forall z \notin \langle zs, x\rangle.\ \Sigma_z\ (v_1\delta)\ (v_1 \mid \delta) = \Sigma_z\ (v_1\delta)\ (\langle s\rangle(\rho, x = \sigma_x\ v_1\ (v_1\uparrow))\delta)$ and (4b) $\forall z \notin \langle zs, x\rangle.\ \sigma_z\ (v_1\delta)\ (v_1 \mid \delta) = \sigma_z\ (v_1\delta)\ (\langle s\rangle(\rho, x = \sigma_x\ v_1\ (v_1\uparrow))\delta)$. Conditions (1) and (3) follow from assumption $y \neq x$ and inversion of $\vdash_\delta v$, which also gives (5a) $\forall z \notin \langle zs, x\rangle.\ \Sigma_z\ (v_1\delta)\ (v_1 \mid \delta) = \Sigma_z\ (v_1\delta)\ (\langle x : a, s\rangle\rho\delta)$ and (5b) $\forall z \notin \langle zs, x\rangle.\ \sigma_z\ (v_1\delta)\ (v_1 \mid \delta) = \sigma_z\ (v_1\delta)\ (\langle x : a, s\rangle\rho\delta)$ and (6) $\forall d.\ (\rho\delta, x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta = d) \vdash \langle s\rangle$. Instantiating (5a) and (5b) with x (using the fact that $x \notin zs$), we obtain (7a) $\Sigma_x\ (v_1\delta)\ (v_1 \mid \delta) = \mathsf{p}_{\mathsf{V}}\, a\rho\delta$ and (7b) $\sigma_x\ (v_1\delta)\ (v_1 \mid \delta) = \mathsf{p}_{\mathsf{p}_{\mathsf{V}}\, a\rho\delta}(x(v_1\delta))$. To establish (4a), consider $z \notin \langle zs, x\rangle$, so that $z \neq x$. Then we have

$$
\begin{array}{ll}
& \Sigma_z\ (v_1\delta)\ (v_1 \mid \delta) \\
= & \hfill \{(5a)\} \\
& \Sigma_z\ (v_1\delta)\ (\langle x : a, s\rangle\rho\delta) \\
= & \hfill \{\text{semantics of expressions}\} \\
& \Sigma_z\ (v_1\delta)\ [x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta, \lambda d.\langle s\rangle(\rho\delta, x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta = d)] \\
= & \hfill \{z \neq x\} \\
& \Sigma_z\ (v_1\delta)\ (\langle s\rangle(\rho\delta, x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta = \mathsf{p}_{\mathsf{p}_{\mathsf{V}}\, a\rho\delta}(x(v_1\delta)) \\
= & \hfill \{\text{(7a) and (7b), semantics of environments}\} \\
& \Sigma_z\ (v_1\delta)\ (\langle s\rangle(\rho, x = \sigma_x\ v_1\ (v_1\uparrow))\delta)
\end{array}
$$

as required. The proof for (4b) is similar. By similar reasoning (but using the assumption $y \neq x$ instead of $z \neq x$) we also obtain the required equality $v\delta = v'\delta$. By definition we have $v \mid \delta = v' \mid \delta$.

- Case $v_0\uparrow$.
  - Subcase v_0 is one of $\star$, $e\rho$, $e \mid \rho$, $v_1\ v_2$, $v_1 * v_2$, $v_1\uparrow$, $\pi_c\ t$, $\sigma_x\ v_1\ v_2$, $\Sigma_x\ v_1\ v_2$, $\Sigma_x^{zs}\ v_1\ s\ \rho$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ hold by definition.
  - Subcase v_0 is one of v_k^t, $\perp\!\!\!\perp^t$, $\gamma\ t\ b\ v\ \rho$. Inversion of $\vdash_\delta v$ immediately gives $\vdash_\delta v'$ and (1) $\vdash_\delta t : \mathsf{V}$. The equalities $v\delta = v'\delta$ and $v \mid \delta = v' \mid \delta$ follow from (1).
  - Subcase v_0 is $\sigma_x\ r\ \rho\ s\ \rho'$. The desired conclusions follow from inversion of $\vdash_\delta v$ and induction hypothesis on v_0.

- Case $(\gamma\ t\ b\ w\ \rho)\ v_0$. For $\vdash_\delta v'$, we need to show (1) $\vdash_\delta (\rho, y = v_0)$, (2) $\vdash_\delta t * v_0 : \mathsf{V}$, (3) $\vdash_\delta w : \mathsf{Data}\ \Delta$ and (4) $\Delta; (\rho, y = v_0)\delta \vdash w\delta; (b\ y) : (t * v_0)\delta$. Inversion of $\vdash_\delta v$ gives (1) and (3), as well as (5) $\vdash_\delta t : \mathsf{V}$, (6) $\mathsf{p_V} t\delta = \mathsf{Prod}\ (v_0 \mid \delta)\ f$ and (7) $\Delta; \rho\delta \vdash w\delta; b : t\delta$. Conditions (5) and (6) imply (2). We prove that (7) implies (4) by induction on b.
  - Case $(\bullet)$. Immediate.
  - Case $(c\ x.e, b)$. Suppose (8) $\Delta; \rho\delta \vdash w\delta; (c\ x.e, b) : t\delta$. We need to show (9) $\Delta; (\rho, y = v_0)\delta \vdash w\delta; (c\ x.(e\ y), (b\ y)) : (t * v_0)\delta$. By inversion of (9), it suffices to show $c\Delta = \mathsf{Def}\ u$, $\Delta; (\rho, y = v_0)\delta \vdash w\delta; (b\ y) : (t * v_0)\delta$ and (10) $\forall d.(w\delta = \mathsf{Inj}_c\ d \implies ((\rho, y = v_0)\delta, x : u = d) \vdash e\ y : (t * v_0)\delta)$. The former two conditions follow immediately from inversion of (8) and induction hypothesis on b. For (10), let $d \in \mathsf{D}$ and suppose $w\delta = \mathsf{Inj}_c\ d$. By (8) we then have (11) $(\rho\delta, x : u = d) \vdash e : t\delta$, and for (10) we need to show (12) $((\rho, y = v_0)\delta, x : u = d) \vdash e\ y : (t * v_0)\delta$. By inversion of (12), it suffices to show (13) $((\rho, y = v_0)\delta, x : u = d) \vdash e$, (14) $e \mid ((\rho, y = v_0)\delta, x : u = d) = \mathsf{Prod}\ (y \mid ((\rho, y = v_0)\delta, x : u = d))\ f$, (15) $((\rho, y = v_0)\delta, x : u = d) \vdash y$ and (16) $(e\ y) \mid ((\rho, y = v_0)\delta, x : u = d) = (t * v_0)\delta$. Condition (15) is immediate, and for (13), (14) and (16), we have

$$
\begin{array}{ll}
 & (\rho\delta, x : u = d) \vdash e \\
\implies & \qquad\qquad \{\text{by an induction on } e, \text{ using } y \notin \textsc{fv}(e)\} \\
 & ((\rho, y = v_0)\delta, x : u = d) \vdash e
\end{array}
$$

and

$$
\begin{array}{ll}
 & e \mid ((\rho, y = v_0)\delta, x : u = d) \\
= & \hfill \{y \notin \textsc{fv}(e)\} \\
 & e \mid (\rho\delta, x : u = d) \\
= & \hfill \{(11)\} \\
 & t\delta \\
= & \hfill \{(5),(6)\} \\
 & \mathsf{Prod}\ (v_0 \mid \delta)\ f \\
= & \hfill \{x \neq y\} \\
 & \mathsf{Prod}\ (y \mid ((\rho, y = v_0)\delta, x : u = d))\ f
\end{array}
$$

and

$$
\begin{array}{ll}
 & (e\ y) \mid ((\rho, y = v_0)\delta, x : u = d) \\
= & \hfill \{\text{semantics of expressions}\} \\
 & (e \mid ((\rho, y = v_0)\delta, x : u = d)) * (y((\rho, y = v_0)\delta, x : u = d)) \\
= & \hfill \{x \neq y\} \\
 & (e \mid ((\rho, y = v_0)\delta, x : u = d)) * v_0\delta \\
= & \hfill \{\text{by an induction on } e, \text{ provided } y \notin \textsc{fv}(e)\} \\
 & (e \mid (\rho\delta, x : u = d)) * v_0\delta \\
= & \hfill \{(11)\} \\
 & t\delta * v_0\delta \\
= & \hfill \{\text{semantics of values}\} \\
 & (t * v_0)\delta
\end{array}
$$

respectively.

We now prove $v\delta = ((\gamma\ t\ b\ w\ \rho)\ v_0)\delta = (\gamma\ (t * v_0)\ (b\ y)\ w\ (\rho, y = v_0))\delta = v'\delta$ by induction on b.

– Case ($\bullet$). By definitions, both sides are equal to $\perp$.

– Case $(c\ x.e, b)$ with $w\delta = \mathsf{Inj}_c\ d$. We have

$$
\begin{array}{ll}
& ((\gamma\ t\ (c\ x.e, b)\ w\ \rho)\ v_0)\delta \\
= & \hfill \{\text{semantics of values}\} \\
& \mathsf{p}_{\mathsf{pv}\,t\delta}(\llbracket c\ x.e, b \rrbracket\ w\delta\ (w \mid \delta)\ \rho\delta) \cdot v_0\delta \\
= & \hfill \{(3),\ \text{assumption } w\delta = \mathsf{Inj}_c\ d\} \\
& \mathsf{p}_{\mathsf{pv}\,t\delta}(\llbracket c\ x.e, b \rrbracket\ (\mathsf{Inj}_c\ d)\ (\mathsf{Data}\ \Delta)\ \rho\delta) \cdot v_0\delta \\
= & \hfill \{c\Delta = \mathsf{Def}\ u \text{ by } (3)\} \\
& \mathsf{p}_{\mathsf{pv}\,t\delta}(e(\rho\delta, x : u = d)) \cdot v_0\delta \\
= & \hfill \{(\rho\delta, x : u = d) \vdash e : t\delta = \mathsf{Prod}\ (v_0 \mid \delta)\ f \text{ by } (5) \text{ and } (8)\} \\
& e(\rho\delta, x : u = d) \cdot v_0\delta \\
= & \hfill \{\text{assumptions } x \neq y \text{ and } y \notin \textsc{fv}(e)\} \\
& e((\rho, y = v_0)\delta, x : u = d) \cdot (y((\rho, y = v_0)\delta, x : u = d)) \\
= & \hfill \{\text{semantics of expressions and environments}\} \\
& (e\ y)((\rho, y = v_0)\delta, x : u = d) \\
= & \hfill \{(2),\ (12)\} \\
& \mathsf{p}_{\mathsf{pv}\,(t * v_0)\delta}(e\ y)((\rho, y = v_0)\delta, x : u = d) \\
= & \hfill \{c\Delta = \mathsf{Def}\ u \text{ by } (3)\} \\
& \mathsf{p}_{\mathsf{pv}\,(t * v_0)\delta}(\llbracket c\ x.(e\ y), (b\ y) \rrbracket\ (\mathsf{Inj}_c\ d)\ (\mathsf{Data}\ \Delta)\ (\rho, y = v_0)\delta) \\
= & \hfill \{(3),\ \text{assumption } w\delta = \mathsf{Inj}_c\ d\} \\
& \mathsf{p}_{\mathsf{pv}\,(t * v_0)\delta}(\llbracket c\ x.e\ y, b\ y \rrbracket\ w\delta\ (w \mid \delta)\ (\rho, y = v_0)\delta) \\
= & \hfill \{\text{semantics of values}\} \\
& (\gamma\ (t * v_0)\ (c\ x.(e\ y), (b\ y))\ w\ (\rho, y = v_0))\delta
\end{array}
$$

– Case $(c\ x.e, b)$ with $w\delta = \mathsf{Inj}_{c'}\ d$, $c' \neq c$. We have

$$
\begin{array}{ll}
& ((\gamma\ t\ (c\ x.e, b)\ w\ \rho)\ v_0)\delta \\
= & \hfill \{\text{semantics of values}\} \\
& \mathsf{p}_{\mathsf{pv}\,t\delta}(\llbracket c\ x.e, b \rrbracket\ w\delta\ (w \mid \delta)\ \rho\delta) \cdot v_0\delta \\
= & \hfill \{w\delta = \mathsf{Inj}_{c'}\ d,\ c' \neq c,\ (3),\ c'\Delta = \mathsf{Def}\ u\} \\
& \mathsf{p}_{\mathsf{pv}\,t\delta}(\llbracket b \rrbracket\ w\delta\ (w \mid \delta)\ \rho\delta) \cdot v_0\delta \\
= & \hfill \{\text{semantics of values}\} \\
& ((\gamma\ t\ b\ w\ \rho)\ v_0)\delta \\
= & \hfill \{\text{induction hypothesis on } b\} \\
& (\gamma\ (t * v_0)\ (b\ y)\ w\ (\rho, y = v_0))\delta
\end{array}
$$

– Case $(c\ x.e, b)$ with $w\delta = \bot$. Both sides are equal to $\bot$, by definitions.

The equation $v \mid \delta = v' \mid \delta$ follows from (2) and (5).

- Cases $(\gamma\ t\ b\ w\ \rho) * v_0$, $\pi_c\ (\gamma\ t\ b\ w\ \rho)$. Similar to previous case.

- Case $\sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v_0$. For $\vdash_\delta v'$, we need to show (1) $\vdash_\delta \rho$, (2) $\vdash_\delta \Sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v_0$, (3) $\vdash_\delta w : \mathsf{Data}\ \Delta$ and (4) $\Delta; \rho\delta \vdash w\delta; (b.x) : (\Sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v_0)\delta$. Inversion of $\vdash_\delta v$ gives (1), (2) and (3), as well as (5) $\Delta; \rho\delta \vdash w\delta; b : t\delta$, (6) $\vdash_\delta \gamma\ t\ b\ w\ \rho : v_0\delta = [xs : s]$, (7) $x \in xs$ and (8) $\vdash_\delta t : \mathsf{V}$. We prove that (5) implies (6) by induction on b.

  - Case $(\bullet)$. Immediate.
  - Case $(c\ y.e, b)$. Suppose $\Delta; \rho\delta \vdash w\delta; (c\ y.e, b) : t\delta$. Inversion gives (9) $\Delta; \rho\delta \vdash w\delta; b : t\delta$, (10) $\forall d.\ (w\delta = \mathsf{Inj}_c\ d \implies (\rho\delta, y : u = d) \vdash e : t\delta)$ and (11) $c\Delta = \mathsf{Def}\ u$. We need to show that $\Delta; \rho\delta \vdash w\delta; (c\ y.(e.x), (b.x)) : (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta$. By inversion, it suffices to show (12) $\Delta; \rho\delta \vdash w\delta; b.x : (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta$ and (13) $\forall d.\ (w\delta = \mathsf{Inj}_c\ d \implies (\rho\delta, y : u = d) \vdash e.x : (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta)$. Condition (9) implies (12) since

$$
\begin{aligned}
& \Delta; \rho\delta \vdash w\delta; b : t\delta \\
\implies & \qquad \{\text{induction hypothesis}\} \\
& \Delta; \rho\delta \vdash w\delta; b.x : (\Sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v_0)\delta \\
\iff & \qquad \{\text{Lemma 17, } c \notin dom(b)\} \\
& (\forall d.\ w\delta \neq \mathsf{Inj}_c\ d \implies \Delta; \rho\delta \vdash w\delta; b.x : (\Sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v_0)\delta) \\
& \qquad \wedge (\forall c' \in dom(b).\ c'\Delta = \mathsf{Def}\ u') \\
\iff & \qquad \{\text{if } w\delta \neq \mathsf{Inj}_c\ d \text{ then } (\gamma\ t\ b\ w\ \rho)\delta = (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\delta\} \\
& (\forall d.\ w\delta \neq \mathsf{Inj}_c\ d \implies \Delta; \rho\delta \vdash w\delta; b.x : (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta) \\
& \qquad \wedge (\forall c' \in dom(b).\ c'\Delta = \mathsf{Def}\ u') \\
\iff & \qquad \{\text{Lemma 17, } c \notin dom(b)\} \\
& \Delta; \rho\delta \vdash w\delta; b.x : (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta
\end{aligned}
$$

  To show that (10) implies (13), suppose $w\delta = \mathsf{Inj}_c\ d$. We need to show that $(\rho\delta, y : u = d) \vdash e.x : (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta$. By inversion, it suffices to show (14) $(\rho\delta, y : u = d) \vdash e$, (15) $e \mid (\rho\delta, y : u = d) = [xs : s]$, (16) $x \in xs$ and (17) $e.x \mid (\rho\delta, y : u = d) = (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta$. Condition (14) follows from (10). For (15) and (16), we have

$$
\begin{array}{ll}
& e \mid (\rho\delta, y : u = d) \\
= & \hfill \{(10),\ (8)\} \\
& \mathsf{p_V}\, t\delta \\
= & \hfill \{\text{semantics of values}\} \\
& \gamma\ t\ b\ w\ \rho \mid \delta \\
= & \hfill \{(6)\} \\
& v_0\delta \\
= & \hfill \{(6)\} \\
& [xs : s]
\end{array}
$$

where $x \in xs$ by (7), as required. We also have (19)

$$
\begin{array}{ll}
& e(\rho\delta, y : u = d) \\
= & \hfill \{(10)\} \\
& \mathsf{p}_{\mathsf{p_V}\, t\delta}\, e(\rho\delta, y : u = d) \\
= & \hfill \{w\delta = \mathsf{Inj}_c\ d,\ w \mid \delta = \mathsf{Data}\ \Delta,\ c\Delta = \mathsf{Def}\ u\} \\
& \mathsf{p}_{\mathsf{p_V}\, t\delta}([\![c\ y.e, b]\!]\ w\delta\ (w \mid \delta)\ \rho\delta) \\
= & \hfill \{\text{semantics of values}\} \\
& (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\delta
\end{array}
$$

which then gives

$$
\begin{array}{ll}
& e.x \mid (\rho\delta, y : u = d) \\
= & \hfill \{\text{semantics of expressions}\} \\
& \Sigma_x\ (e(\rho\delta, y : u = d))\ (e \mid (\rho\delta, y : u = d)) \\
= & \hfill \{(6)\} \\
& \Sigma_x\ (e(\rho\delta, y : u = d))\ v_0\delta \\
= & \hfill \{(19)\} \\
& (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta
\end{array}
$$

as required for (17).

We prove $v\delta = v'\delta$ by induction on b.

- Case ($\bullet$). Immediate.

– Case $(c\ y.e, b)$ with $w\delta = \mathsf{Inj}_c\ d$.

$$
\begin{array}{ll}
& (\sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta \\
= & \hfill \{\text{semantics of values}\} \\
& \sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\delta\ (v_0\delta) \\
= & \hfill \{w\delta = \mathsf{Inj}_c\ d,\ (19),\ (6),\ (15)\} \\
& \Sigma_x\ (e(\rho\delta, y : u = d))\ (e \mid (\rho\delta, y : u = d)) \\
= & \hfill \{\text{semantics of expressions}\} \\
& (e.x)(\rho\delta, y : u = d) \\
= & \hfill \{\text{Lemma 1}\} \\
& \mathsf{p}_{(e.x)\mid(\rho\delta, y:u=d)}(e.x)(\rho\delta, y : u = d) \\
= & \hfill \{w\delta = \mathsf{Inj}_c\ d,\ w \mid \delta = \mathsf{Data}\ \Delta,\ c\Delta = \mathsf{Def}\ u,\ (17)\} \\
& \mathsf{p}_{\mathsf{pV}(\Sigma_x\ (\gamma\ t\ (c\ y.e,b)\ w\ \rho)\ v_0)\delta}([\![c\ y.(e.x), (b.x)]\!]\ w\delta\ (w \mid \delta)\ \rho\delta) \\
= & \hfill \{\text{semantics of values}\} \\
& (\gamma\ (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\ (c\ y.(e.x), (b.x))\ w\ \rho)\delta
\end{array}
$$

– Case $(c\ y.e, b)$ with $w\delta = \mathsf{Inj}_{c'}\ d$ and $c' \neq c$.

$$
\begin{array}{ll}
& (\sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\delta \\
= & \hfill \{\text{semantics of values}\} \\
& \sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\delta\ (v_0\delta) \\
= & \hfill \{w\delta = \mathsf{Inj}_{c'}\ d,\ c' \neq c\} \\
& \sigma_x\ (\gamma\ t\ b\ w\ \rho)\delta\ (v_0\delta) \\
= & \hfill \{\text{induction hypothesis}\} \\
& (\gamma\ (\Sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v_0)\ (b.x)\ w\ \rho)\delta \\
= & \hfill \{\text{semantics of values}\} \\
& \mathsf{p}_{\mathsf{pV}(\Sigma_x\ (\gamma\ t\ b\ w\ \rho)\ v_0)\delta}([\![b.x]\!]\ w\delta\ (w \mid \delta)\ \rho\delta) \\
= & \hfill \{w\delta = \mathsf{Inj}_{c'}\ d,\ c' \neq c\} \\
& \mathsf{p}_{\mathsf{pV}(\Sigma_x\ (\gamma\ t\ (c\ y.e,b)\ w\ \rho)\ v_0)\delta}([\![(c\ y.(e.x), (b.x))]\!]\ w\delta\ (w \mid \delta)\ \rho\delta) \\
= & \hfill \{\text{semantics of values}\} \\
& (\gamma\ (\Sigma_x\ (\gamma\ t\ (c\ y.e, b)\ w\ \rho)\ v_0)\ (c\ y.(e.x), (b.x))\ w\ \rho)\delta
\end{array}
$$

– Case $(c\ y.e, b)$ with $w\delta = \bot$. Both sides are equal to $\bot$.

The equation $v \mid \delta = v' \mid \delta$ follows from the semantics of values.

- Cases $\sigma_x\ v_0\ (\gamma\ t\ b\ w\ \rho)$, $\Sigma_x\ v_0\ (\gamma\ t\ b\ w\ \rho)$. Similar to previous case.
- Other forms of v cannot stand in relation $v \longrightarrow\!\!\triangleright\ v'$.

□

B.6 Proof of Proposition 4

Let v_1, v_2 be values. Then for all valuations δ,

$$\vdash_\delta v_i \ \wedge\ v_1 \approx v_2 \ \wedge\ v_1 \mid \delta = v_2 \mid \delta \implies v_1\delta = v_2\delta.$$

$$\vdash_\delta w_i \ \wedge\ w_1 \approx w_2 \implies w_1\delta = w_2\delta \ \wedge\ w_1 \mid \delta = w_2 \mid \delta.$$

Proof. By induction on the derivation of $v_1 \approx v_2$.

- Case v_1 or v_2 not in weak head normal form. Then Proposition 3 applies to $v_1 \longrightarrow\!\!\triangleright^* h_1$ and $v_2 \longrightarrow\!\!\triangleright^* h_2$, giving $\vdash_\delta h_i$, $v_i\delta = h_i\delta$ and $v_i \mid \delta = h_i \mid \delta$, for $i \in \{1,2\}$. By assumption $v_1 \mid \delta = v_2 \mid \delta$, induction hypothesis now applies to $h_1 \approx h_2$, giving $v_1\delta = h_1\delta = h_2\delta = v_2\delta$.

- Cases $\star \approx \star$, $\perp\!\!\!\perp^{t_1} \approx \perp\!\!\!\perp^{t_2}$. Immediate.

- Case $\lambda x_1 : a_1.e_1\rho_1 \approx \lambda x_2 : a_2.e_2\rho_2$. Inversion of $\vdash_\delta \lambda x_i : a_i.e_i\rho_i$ gives (1) $\vdash_\delta \rho_i$, (2) $\vdash_\delta \rho_i\delta \vdash a_i : \mathsf{V}$ and (3) $\forall d.\ (\rho_i\delta, x_i : \mathsf{p_V}\, a_i\rho_i\delta = d) \vdash e_i$. Since Prod is injective, the assumption $\lambda x_1 : a_1.e_1\rho_1 \mid \delta = \lambda x_2 : a_2.e_2\rho_2 \mid \delta$ combined with (2) implies (4) $a_1\rho_1\delta = \mathsf{p_V}\, a_1\rho_1\delta = \mathsf{p_V}\, a_2\rho_2\delta = a_2\rho_2\delta$, for which we write u, and (5) $\forall d.\ e_1 \mid (\rho_1\delta, x_1 : u = d) = e_2 \mid (\rho_2\delta, x_2 : u = d)$. We need to show $\lambda x_1 : a_1.e_1\rho_1\delta = \lambda x_2 : a_2.e_2\rho_2\delta$, that is $\forall d.\ e_1(\rho_1\delta, x_1 : u = d) = e_2(\rho_2\delta, x_2 : u = d)$. Let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. Freshness of $\mathrm{v}_k^{a_i\rho_i}$ implies that $\vdash_\delta \rho_i \iff \vdash_{\delta'} \rho_i$ and $\rho_i\delta = \rho_i\delta'$. Hence we have $\vdash_{\delta'} e_i(\rho_i, x_i = \mathrm{v}_k^{a_i\rho_i})$ by (1–3) and freshness of $\mathrm{v}_k^{a_i\rho_i}$, and we have $e_1\rho_1' \mid \delta' = e_2\rho_2' \mid \delta'$ by (5) and freshness of $\mathrm{v}_k^{a_i\rho_i}$. Now induction hypothesis applies to $e_1\rho_1' \approx e_2\rho_2'$, giving $e_1(\rho_1\delta, x_1 : u = d) = e_1\rho_1'\delta' = e_2\rho_2'\delta' = e_2(\rho_2\delta, x_2 : u = d)$ as desired.

- Cases $\lambda x_1 : a_1.e_1 \mid \rho_1 \approx \lambda x_2 : a_2.e_2 \mid \rho_2$, $\lambda x_1 : a_1.e_1 \mid \rho_1 \approx \forall x_2 : a_2.a_4\rho_2$, $\forall x_1 : a_1.a_3\rho_1 \approx \lambda x_2 : a_2.e_2 \mid \rho_2$, $\forall x_1 : a_1.a_3\rho_1 \approx \forall x_2 : a_2.a_4\rho_2$. Similar to previous case.

- Case $\langle\bullet\rangle\rho_1 \approx \langle\bullet\rangle\rho_2$. Immediate.

- Case $\langle x : a_1, s_1\rangle\rho_1 \approx \langle x : a_2, s_2\rangle\rho_2$. We need to show $\langle x : a_1, s_1\rangle\rho_1\delta = \langle x : a_2, s_2\rangle\rho_2\delta$, that is (1) $\mathsf{p_V}\, a_1\rho_1\delta = \mathsf{p_V}\, a_2\rho_2\delta$ and (2) $\lambda d.\langle s_1\rangle(\rho_1\delta, x : \mathsf{p_V}\, a_1\rho_1\delta = d) = \lambda d.\langle s_2\rangle(\rho_2\delta, x : \mathsf{p_V}\, a_2\rho_2\delta = d)$. By inversion of $\vdash_\delta \langle x : a_i, s_i\rangle\rho_i$, we have (3) $\vdash_\delta \rho_i$, (4) $\rho_i\delta \vdash a_i : \mathsf{V}$ and (5) $\forall d.\ (\rho_i\delta, x : \mathsf{p_V}\, a_i\rho_i\delta = d) \vdash \langle s_i\rangle$. Thus $\vdash_\delta a_i\rho_i$ by (3) and (4), and $a_1\rho_1 \mid \delta = a_2\rho_2 \mid \delta$ by (4). Now induction hypothesis applies to $\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2$, giving $\mathsf{p_V}\, a_1\rho_1\delta = a_1\rho_1\delta = a_2\rho_2\delta = \mathsf{p_V}\, a_2\rho_2\delta$ by (4), as required for (1). Let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. We have $\vdash_{\delta'} \langle s_i\rangle(\rho_i, x = \mathrm{v}_k^{a_i\rho_i})$ by (3–5) and by freshness of $\mathrm{v}_k^{a_i\rho_i}$. Hence induction hypothesis applies to $\langle s_1\rangle(\rho_1, x = \mathrm{v}_k^{a_1\rho_1}) \approx \langle s_2\rangle(\rho_2, x = \mathrm{v}_k^{a_2\rho_2})$, giving $\forall d.\ \langle s_1\rangle(\rho_1\delta, x : \mathsf{p_V}\, a_1\rho_1\delta = d) = \langle s_2\rangle(\rho_2\delta, x : \mathsf{p_V}\, a_2\rho_2\delta = d)$, as required for (2).

- Case $\langle x = e_1, s_1\rangle\rho_1 \approx \langle x = e_2, s_2\rangle\rho_2$. Similar to previous case.

- Case $\langle r_1 :: a_1\rangle\rho_1 \approx \langle r_2 :: a_2\rangle\rho_2$. We need to show $\langle r_1 :: a_1\rangle\rho_1\delta = \tau\ (\mathsf{p_V}\, a_1\rho_1\delta)\ (r_1\rho_1\delta) = \tau\ (\mathsf{p_V}\, a_2\rho_2\delta)\ (r_2\rho_2\delta) = \langle r_2 :: a_2\rangle\rho_2\delta$. It thus suffices to show (1) $\mathsf{p_V}\, a_1\rho_1\delta = \mathsf{p_V}\, a_2\rho_2\delta$ and (2) $r_1\rho_1\delta = r_2\rho_2\delta$. Condition (1) holds by assumption $\langle r_1 :: a_1\rangle\rho_1 \mid \delta = \langle r_2 :: a_2\rangle\rho_2 \mid \delta$. By inversion of $\vdash_\delta \langle r_i :: a_i\rangle\rho_i$, we have (3) $\vdash_\delta \rho_i$, (4) $\rho_i\delta \vdash a_i : \mathsf{V}$, (5) $a_i\rho_i\delta = [xs_i : s_i]$ and (6) $\rho_i\delta \models r_i : [xs_i : s_i]$. Conditions (1), (4) and (5) imply that (7) $[xs_1 : s_1] = [xs_2 : s_2]$. By assumption, we have (8) $\vdash_\delta \phi$ and (9) $\vdash_\delta \theta$. Inversion of $\langle r_1 :: a_1\rangle\rho_1 \approx \langle r_2 :: a_2\rangle\rho_2$ gives (10) $r_1\rho_1 \sim r_2\rho_2$. We now show that (3) and (6–10) imply (2), by induction on $xs_1 = xs_2$.
  - Case $\langle\rangle$. Since $\rho_i\delta \models r_i : []$, we must have $r_i = ()$. Thus (2) is immediate.
  - Case $\langle x^-, xs\rangle$. Then $[xs_i : s_i] = [x : u_i, xs : f_i]$. By (6), we have $\rho_i\delta \models r_i : [x : u_i, xs : f_i]$, where r_i must be of the form $(x = e_i, r_i')$. By inversion of (6), we thus obtain (11) $\rho_i\delta \vdash e_i : u_i$ and (12) $\rho_i\delta \models r_i' : [xs : f_i(e_i\rho_i\delta)]$. Furthermore, inversion of (10) gives (13) $e_1\rho_1 \approx e_2\rho_2$ and (14) $r_1'\rho_1 \sim r_2'\rho_2$. Now (7) and (11) imply that (15) $e_1 \mid \rho_1\delta = e_2 \mid \rho_2\delta$. By (3), (8), (9), (11) and (15), induction hypothesis applies to (13), giving (16) $e_1\rho_1\delta = e_2\rho_2\delta$. By (7) and (16), we have (17) $f_1(e_1\rho_1\delta) = f_2(e_2\rho_2\delta)$. By (3), (8), (9), (12) and (17), induction hypothesis applies to (14), giving (18) $r_1'\rho_1\delta = r_2'\rho_2\delta$. Now conditions (16) and (18) together give (2), as required.
  - Case $\langle x^+, xs\rangle$. By (6), we have $\rho_i\delta \models r_i : [x : u_i = d_i, xs : s_i']$, and by inversion we obtain $\rho_i\delta \models r_i : [xs : s_i']$. By (7), we have $s_1' = s_2'$. The desired result now follows immediately by induction hypothesis.
- Case $\mathrm{v}_k^{t_1} \approx \mathrm{v}_k^{t_2}$. By inversion of $\mathrm{v}_k^{t_i}$, we have $\vdash_\delta t_i : \mathsf{V}$. Now induction hypothesis applies to $t_1 \approx t_2$, giving $t_1\delta = t_2\delta$. Thus for $\delta(k) = d$, we have $\mathrm{v}_k^{t_1} \mid \delta = \mathsf{p_V}\, t_1\delta = \mathsf{p_V}\, t_2\delta = \mathrm{v}_k^{t_2} \mid \delta$ and $\mathrm{v}_k^{t_1}\delta = \mathsf{p}_{\mathsf{p_V}\, t_1\delta}\, d = \mathsf{p}_{\mathsf{p_V}\, t_2\delta}\, d = \mathrm{v}_k^{t_2}\delta$, as required.
- Case $w_1\ v_1 \approx w_2\ v_2$. By inversion of $\vdash_\delta w_i\ v_i$, we have (1) $\vdash_\delta w_i$, (2) $w_i \mid \delta = \mathsf{Prod}\ (v_i \mid \delta)\ f_i$ and (3) $\vdash_\delta v_i$. Induction hypothesis on $w_1 \approx w_2$ gives (4) $w_1\delta = w_2\delta$ and (5) $w_1 \mid \delta = w_2 \mid \delta$. By (2), (5) and injectivity of Prod , we then have (6) $v_1 \mid \delta = v_2 \mid \delta$. By (3) and (6), induction hypothesis applies to $v_1 \approx v_2$, giving $v_1\delta = v_2\delta$. Hence by (4–6) we have $(w_1\ v_1)\delta = w_1\delta \cdot v_1\delta = w_2\delta \cdot v_2\delta = (w_2\ v_2)\delta$ and $(w_1\ v_1) \mid \delta = (w_1 \mid \delta) * v_1\delta = (w_2 \mid \delta) * v_2\delta = (w_2\ v_2) \mid \delta$, as required.
- Case $w_1 * v_1 \approx w_1 * v_2$. Similar to previous case.
- Case $\pi_c\ w_1 \approx \pi_c\ w_2$. The desired conlusions follow from inversion of $\vdash_\delta \pi_c\ w_i$ and induction hypothesis on $w_1 \approx w_2$.
- Case $\mu x_1 : a_1.e_1\rho_1 \approx \mu x_2 : a_2.e_2\rho_2$. Inversion of $\vdash_\delta \mu x_i : a_i.e_i\rho_i$ gives (1) $\vdash_\delta \rho_i$, (2) $\rho_i\delta \vdash a_i : \mathsf{V}$ and (3) $\forall d.\ (\rho_i\delta, x_i : \mathsf{p_V}\, a_i\rho_i\delta = d) \vdash e_i : a_i\rho_i\delta$. Thus $\vdash_\delta a_i\rho_i : \mathsf{V}$ by (1) and (2), so that induction hypothesis applies to $a_1\rho_1 \approx a_2\rho_2$, giving (4) $\mathsf{p_V}\, a_1\rho_1\delta = a_1\rho_1\delta = a_2\rho_2\delta = \mathsf{p_V}\, a_2\rho_2\delta$. By (3) and (4), we have $\forall d.\ e_1 \mid (\rho_1\delta, x_1 : \mathsf{p_V}\, a_1\rho_1\delta = d) = e_2 \mid (\rho_2\delta, x_2 : \mathsf{p_V}\, a_2\rho_2\delta = d)$. We need to show $\mu x_1 : a_1.e_1\rho_1\delta = \mu x_2 : a_2.e_2\rho_2\delta$, where $\mu x_i : a_i.e_i\rho_i\delta = \mathsf{fix}\,(\lambda d.\mathsf{p}_{\mathsf{p_V}\, a_i\rho_i\delta}\, e_i(\rho_i\delta, x_i : \mathsf{p_V}\, a_i\rho_i\delta = d))$. By (2) and (3), it

suffices to show $\forall d.\ e_1(\rho_1\delta, x_1 : \mathsf{p_V}\, a_1\rho_1\delta = d) = e_2(\rho_2\delta, x_2 : \mathsf{p_V}\, a_2\rho_2\delta = d)$. Let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. Freshness of $\mathrm{v}_k^{a_i\rho_i}$ implies that $\vdash_\delta \rho_i \iff \vdash_{\delta'} \rho_i$ and $\rho_i\delta = \rho_i\delta'$. Hence we have $\vdash_{\delta'} e_i(\rho_i, x_i = \mathrm{v}_k^{a_i\rho_i})$ by (1–3) and freshness of $\mathrm{v}_k^{a_i\rho_i}$, and we have $e_1\rho_1' \mid \delta' = e_2\rho_2' \mid \delta'$ by (4) and freshness of $\mathrm{v}_k^{a_i\rho_i}$. Now induction hypothesis applies to $e_1\rho_1' \approx e_2\rho_2'$, giving $e_1(\rho_1\delta, x_1 : u = d) = e_1\rho_1'\delta' = e_2\rho_2'\delta' = e_2(\rho_2\delta, x_2 : u = d)$ as desired.

- Case $\sigma_x\ v_1\ t_1 \approx \sigma_x\ v_2\ t_2$. We need to show that $(\sigma_x\ v_1\ t_1)\delta = (\sigma_x\ v_2\ t_2)\delta$ and $(\sigma_x\ v_1\ t_1) \mid \delta = (\sigma_x\ v_2\ t_2) \mid \delta$. Inversion of $\vdash_\delta \sigma_x\ v_i\ t_i$ gives (1) $\vdash_\delta t_i$, (2) $\vdash_\delta v_i$ and (3) $v_i \mid \delta = t_i\delta = [xs_i : s_i]$ where $x \in xs_i$. By (1) and (3), induction hypothesis applies to $t_1 \approx t_2$ giving (4) $t_1\delta = t_2\delta$. By (2) and (3), we thus have $\vdash_\delta v_i$ and $v_1 \mid \delta = v_2 \mid \delta$, so that induction hypothesis now applies to $v_1 \approx v_2$, giving (5) $v_1\delta = v_2\delta$. Hence by (4) and (5) we have $(\sigma_x\ v_1\ t_1)\delta = \sigma_x\ (v_1\delta)\ (t_1\delta) = \sigma_x\ (v_1\delta)\ (t_1\delta) = (\sigma_x\ v_2\ t_2)\delta$ and $(\sigma_x\ v_1\ t_1) \mid \delta = \Sigma_x\ (v_1\delta)\ (t_1\delta) = \Sigma_x\ (v_1\delta)\ (t_1\delta) = (\sigma_x\ v_2\ t_2) \mid \delta$, as required.
- Case $\Sigma_x\ v_1\ t_1 \approx \Sigma_x\ v_2\ t_2$. Similar to the previous case.
- Case $(c\ e_1\ a_1)\rho_1 \approx (c\ e_2\ a_2)\rho_2$. Inversion of $\vdash_\delta (c\ e_i\ a_i)\rho_i$ gives (1) $\vdash_\delta \rho_i$, (2) $\rho_i\delta \vdash a_i : \mathsf{V}$, (3) $a_i\rho_i\delta = \mathsf{Data}\ \Delta_i$, (4) $c\Delta_i = \mathsf{Def}\ u_i$ and (5) $\rho_i\delta \vdash e_i : u_i$. We need to show $(c\ e_1\ a_1)\rho_1\delta = \mathsf{p}_{\mathsf{p_V} a_1\rho_1\delta}(\mathsf{Inj}_c\ (e_1\rho_1\delta)) = \mathsf{p}_{\mathsf{p_V} a_2\rho_2\delta}(\mathsf{Inj}_c\ (e_2\rho_2\delta)) = (c\ e_2\ a_2)\rho_2\delta$. Assumption $(c\ e_1\ a_1)\rho_1 \mid \delta = (c\ e_2\ a_2)\rho_2 \mid \delta$ gives (6) $a_1\rho_1\delta = \mathsf{p_V}\, a_1\rho_1\delta = \mathsf{p_V}\, a_2\rho_2\delta = a_2\rho_2\delta$, by (1) and (2). It thus suffices to show (7) $e_1\rho_1\delta = e_2\rho_2\delta$. By (3), (6) and injectivity of Data , we have $\Delta_1 = \Delta_2$. By (4) and (5), we then obtain $e_1\rho_1 \mid \delta = e_2\rho_2 \mid \delta$. Since we also have $\vdash_\delta e_i\rho_i$ by (1) and (5), induction hypothesis now applies to $e_1\rho_1 \approx e_2\rho_2$, giving (7) as required.
- Case $\{d_1\}\rho_1 \approx \{d_2\}\rho_2$. We need to show $\{d_1\}\rho_1\delta = \mathsf{Data}\ \Delta_1 = \mathsf{Data}\ \Delta_2 = \{d_2\}\rho_2\delta$, where $\Delta_i = \mathit{data}\ d_i\ (\rho_i\delta)$. Since $c\Delta_i = \bot$ for any $c \notin \mathit{dom}(d_1) = \mathit{dom}(d_2)$, it suffices to show $\forall c \in \mathit{dom}(d_1).\ c\Delta_1 = c\Delta_2$, that is $\forall c \in \mathit{dom}(d_1).\ \pi_c(\{d_1\}\rho_1\delta) = \pi_c(\{d_2\}\rho_2\delta)$. The latter follows from induction hypothesis on the premise $\pi_c(\{d_1\}\rho_1) \approx \pi_c(\{d_2\}\rho_2)$.
- Case $\gamma\ t_1\ b_1\ w_1\ \rho_1 \approx \gamma\ t_2\ b_2\ w_2\ \rho_2$. We need to show $(\gamma\ t_1\ b_1\ w_1\ \rho_1)\delta = (\gamma\ t_2\ b_2\ w_2\ \rho_2)\delta$. By inversion of $\vdash_\delta \gamma\ t_i\ b_i\ w_i\ \rho_i$, we have (1) $\vdash_\delta \rho_i$, (2) $\vdash_\delta w_i$, (3) $w_i \mid \delta = \mathsf{Data}\ \Delta_i$, (4) $\vdash_\delta t_i : \mathsf{V}$ and (5) $\Delta_i; \rho_i\delta \vdash w_i\delta; b_i : t_i\delta$. By assumption and (4), we have (6) $t_1\delta = \mathsf{p_V}\, t_1\delta = \mathsf{p_V}\, t_2\delta = t_2\delta$. Now induction hypothesis on $w_1 \approx w_2$ gives (7) $w_1\delta = w_2\delta$ and (8) $w_1 \mid \delta = w_2 \mid \delta$. Proposition 3 applied to $w{\uparrow} \xrightarrow{\mu}\!\!{\triangleright}^* \{d\}\rho'$ gives (9) $\vdash_\delta \{d\}\rho'$ and (10) $w_1 \mid \delta = \{d\}\rho'\delta = \mathsf{Data}\ \Delta = w_2 \mid \delta$. Using (1–10), we now show that $\vdash_\delta \gamma\ t_i\ b_i\ w_i\ \rho_i$ and $\gamma\ t_1\ b_1\ w_1\ \rho_1 \approx \gamma\ t_2\ b_2\ w_2\ \rho_2$ imply $(\gamma\ t_1\ b_1\ w_1\ \rho_1)\delta = (\gamma\ t_2\ b_2\ w_2\ \rho_2)\delta$ by induction on b_1.
  - Case $b_1 = (\bullet)$. Immediate.
  - Case $b_1 = (c\ x_1.e_1, b_1')$. We must have $b_2 = (c\ x_2.e_2, b_2')$ since $\{d\}\rho' \vdash \gamma\ t_1\ b_1\ w_1\ \rho_1 \approx \gamma\ t_2\ b_2\ w_2\ \rho_2$, and by inversion of the latter we have (11) $e_1(\rho_1, x_1 = \mathrm{v}_k^{\pi_c\ (\{d\}\rho')}) \approx e_2(\rho_2, x_2 = \mathrm{v}_k^{\pi_c\ (\{d\}\rho')})$ and (12) $\{d\}\rho' \vdash \gamma\ t_1\ b_1'\ w_1\ \rho_1 \approx$

$\gamma\ t_2\ b'_2\ w_2\ \rho_2$. By inversion of (5), we have (13) $\forall d'.(w_i\delta = \mathsf{Inj}_c\ d' \implies (\rho_i\delta, x_i : u = d') \vdash e_i : t_i\delta)$, (14) $c\Delta = \mathsf{Def}\ u$ and (15) $\Delta_i; \rho_i\delta \vdash w_i\delta; b'_i : t_i\delta$. Since $w_1\delta = w_2\delta : \mathsf{Data}\ \Delta$, we have either $w_i\delta = \bot$, or, for some $d' \in \mathsf{D}$, $w_i\delta = \mathsf{Inj}_c\ d'$ or $w_i\delta = \mathsf{Inj}_{c'}\ d'$, where $c \neq c'$.

* Subcase $w_i\delta = \bot$. Immediate.
* Subcase $w_i\delta = \mathsf{Inj}_c\ d'$. Consider $\delta' = (\delta, k = d')$. By (1), (9), (10), (13), (14) and freshness of $\mathrm{v}_k^{\pi_c\ (\{d\}\rho')}$, we obtain $\vdash_{\delta'} e_i(\rho_i, x_i = \mathrm{v}_k^{\pi_c\ (\{d\}\rho')})$. By (10), (13), (14) and freshness of $\mathrm{v}_k^{\pi_c\ (\{d\}\rho')}$, we also have $e_1 \mid (\rho_1, x_1 = \mathrm{v}_k^{\pi_c\ (\{d\}\rho')})\delta' = t_i\delta = e_2 \mid (\rho_2, x_2 = \mathrm{v}_k^{\pi_c\ (\{d\}\rho')})\delta'$. Now induction hypothesis applies to (11), giving $e_1(\rho_1, x_1 = \mathrm{v}_k^{\pi_c\ (\{d\}\rho')})\delta' = e_2(\rho_2, x_2 = \mathrm{v}_k^{\pi_c\ (\{d\}\rho')})\delta'$. We also have

$$\begin{array}{ll} & e_i(\rho_i, x_i = \mathrm{v}_k^{\pi_c\ (\{d\}\rho')}) \\ = & \quad \{\text{semantics of environments, freshness of } \mathrm{v}_k^{\pi_c\ (\{d\}\rho')}\text{, (10), (14)}\} \\ & e_i(\rho_i\delta, x_i : u = d') \\ = & \hfill \{(14)\} \\ & \mathsf{p}_{\mathsf{pv}\,t_i\delta}([\![c\ x_i.e_i, b'_i]\!]\ (\mathsf{Inj}_c\ d')\ (\mathsf{Data}\ \Delta)\ \rho_i\delta) \\ = & \hfill \{\text{assumption } w_i\delta = \mathsf{Inj}_c\ d'\text{, (8), (10)}\} \\ & \mathsf{p}_{\mathsf{pv}\,t_i\delta}([\![c\ x_i.e_i, b'_i]\!]\ (w_i\delta)\ (w_i \mid \delta)\ \rho_i\delta) \\ = & \hfill \{\text{semantics of values}\} \\ & (\gamma\ t_i\ (c\ x_i.e_i, b'_i)\ w_i\ \rho_i)\delta \end{array}$$

  as required.
* Subcase $w_i\delta = \mathsf{Inj}_{c'}\ d'$, where $c \neq c'$. By (1–4) and (15), we have $\vdash_\delta \gamma\ t_i\ b'_i\ w_i\ \rho_i$, and by (12) we have $\gamma\ t_1\ b'_1\ w_1\ \rho_1 \approx \gamma\ t_2\ b'_2\ w_2\ \rho_2$, so that induction hypothesis on b'_1 gives $(\gamma\ t_1\ b'_1\ w_1\ \rho_1)\delta = (\gamma\ t_2\ b'_2\ w_2\ \rho_2)\delta$. We also have

$$\begin{array}{ll} & (\gamma\ t_i\ (c'\ x_i.e_i, b'_i)\ w_i\ \rho_i)\delta \\ = & \hfill \{\text{semantics of values}\} \\ & \mathsf{p}_{\mathsf{pv}\,t_i\delta}([\![c'\ x_i.e_i, b'_i]\!]\ (w_i\delta)\ (w_i \mid \delta)\ \rho_i\delta) \\ = & \hfill \{\text{assumptions } w_i\delta = \mathsf{Inj}_{c'}\ d' \text{ and } c \neq c'\text{, (10)}\} \\ & \mathsf{p}_{\mathsf{pv}\,t_i\delta}([\![b'_i]\!]\ (w_i\delta)\ (w_i \mid \delta)\ \rho_i\delta) \\ = & \hfill \{\text{semantics of values}\} \\ & (\gamma\ t_i\ b'_i\ w_i\ \rho_i)\delta \end{array}$$

  as required.

$\square$

B.7 Proof of Proposition 5

Let v, v' be values, θ constraints. Then for all valuations δ,

$$\begin{pmatrix} \vdash_\delta \theta \quad \vdash_\delta v \\ v \xrightarrow{\mu}\!\!\triangleright_\theta v' \end{pmatrix} \implies \begin{pmatrix} \vdash_\delta v' \quad v\delta = v'\delta \\ v \mid \delta = v' \mid \delta \end{pmatrix}.$$

Proof. Proposition 3 has already established the result for all rules except for

$$\frac{(w' \approx v) \in \theta \qquad w \approx w'}{w \longrightarrow\!\!\triangleright_\theta v}$$

where we have $\vdash_\delta w'$ by assumption $\vdash_\delta \theta$. Thus Proposition 4 applies to $w \approx w'$, giving $w\delta = w'\delta$ and $w \mid \delta = w' \mid \delta$. By assumption $\vdash_\delta \theta$, we also have $\vdash_\delta v$, $w'\delta = v\delta$ and $w' \mid \delta = v \mid \delta$, as desired. ☐

B.8 Proof of Proposition 6

Let v_1, v_2 be values. Then for all valuations δ,

$$\begin{pmatrix} \vdash_\delta \phi \quad \vdash_\delta \theta \quad \vdash_\delta v_i \\ \phi; \theta \vdash v_1 \preccurlyeq v_2 \quad v_1 \mid \delta \sqsubseteq v_2 \mid \delta \end{pmatrix} \implies v_1\delta \sqsubseteq v_2\delta.$$

$$\begin{pmatrix} \vdash_\delta \phi \quad \vdash_\delta \theta \quad \vdash_\delta w_i \\ \phi; \theta \vdash w_1 \preccurlyeq w_2 \end{pmatrix} \implies \begin{pmatrix} w_1\delta \sqsubseteq w_2\delta \\ w_1 \mid \delta \sqsubseteq w_2 \mid \delta \end{pmatrix}.$$

Proof. By induction on the derivation of $v_1 \approx v_2$.

- Case v_1 or v_2 not in weak head normal form. Then Proposition 5 applies to $v_1 \longrightarrow\!\!\triangleright_\theta^* h_1$ and $v_2 \longrightarrow\!\!\triangleright_\theta^* h_2$, giving $\vdash_\delta h_i$, $v_i\delta = h_i\delta$ and $v_i \mid \delta = h_i \mid \delta$, for $i \in \{1, 2\}$. By assumption $v_1 \mid \delta = v_2 \mid \delta$, induction hypothesis now applies to $\phi; \theta \vdash h_1 \preccurlyeq h_2$, giving $v_1\delta = h_1\delta = h_2\delta = v_2\delta$.

- Cases $\phi; \theta \vdash \star \preccurlyeq \star$, $\phi; \theta \vdash \perp\!\!\!\perp^t \preccurlyeq h$. Immediate.

- Case $\phi; \theta \vdash \lambda x_1 : a_1.e_1\rho_1 \preccurlyeq \lambda x_2 : a_2.e_2\rho_2$. Inversion of $\vdash_\delta \lambda x_i : a_i.e_i\rho_i$ gives (1) $\vdash_\delta \rho_i$, (2) $\vdash_\delta \rho_i\delta \vdash a_i : \mathsf{V}$ and (3) $\forall d.\ (\rho_i\delta, x_i : \mathsf{p_V}\, a_i\rho_i\delta = d) \vdash e_i$. Since Prod is injective, the assumption $\lambda x_1 : a_1.e_1\rho_1 \mid \delta \sqsubseteq \lambda x_2 : a_2.e_2\rho_2 \mid \delta$ combined with (2) implies (4) $a_1\rho_1\delta = \mathsf{p_V}\, a_1\rho_1\delta \sqsubseteq \mathsf{p_V}\, a_2\rho_2\delta = a_2\rho_2\delta$, and (5) $\forall d.\ e_1 \mid (\rho_1\delta, x_1 : \mathsf{p_V}\, a_1\rho_1\delta = d) \sqsubseteq e_2 \mid (\rho_2\delta, x_2 : \mathsf{p_V}\, a_2\rho_2\delta = d)$. We need to show $\lambda x_1 : a_1.e_1\rho_1\delta \sqsubseteq \lambda x_2 : a_2.e_2\rho_2\delta$, that is $\forall d.\ e_1(\rho_1\delta, x_1 : \mathsf{p_V}\, a_1\rho_1\delta = d) \sqsubseteq e_2(\rho_2\delta, x_2 : \mathsf{p_V}\, a_2\rho_2\delta = d)$. Let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. Freshness of $\mathrm{v}_k^{a_i\rho_i}$ implies that $\vdash_\delta \rho_i \iff \vdash_{\delta'} \rho_i$ and $\rho_i\delta = \rho_i\delta'$. Hence we have $\vdash_{\delta'} e_i(\rho_i, x_i = \mathrm{v}_k^{a_i\rho_i})$ by (1–3) and freshness of $\mathrm{v}_k^{a_i\rho_i}$, and we have $e_1\rho_1' \mid \delta' \sqsubseteq e_2\rho_2' \mid \delta'$ by (5) and freshness of $\mathrm{v}_k^{a_i\rho_i}$. Now induction hypothesis applies to $\phi; \theta \vdash e_1\rho_1' \preccurlyeq e_2\rho_2'$, giving $e_1(\rho_1\delta, x_1 : u = d) = e_1\rho_1'\delta' \sqsubseteq e_2\rho_2'\delta' = e_2(\rho_2\delta, x_2 : u = d)$ as desired.

- Cases $\phi;\theta \vdash \forall x_1 : a_1.a_3\ \rho_1 \preccurlyeq \forall x_2 : a_2.a_4\ \rho_2$, $\phi;\theta \vdash \lambda x_1 : a_1.e_1 \mid \rho_1 \preccurlyeq \lambda x_2 : a_2.e_2 \mid \rho_2$, $\phi;\theta \vdash \forall x_1 : a_1.a_3\ \rho_1 \preccurlyeq \lambda x_2 : a_2.e_2 \mid \rho_2$, $\phi;\theta \vdash \lambda x_1 : a_1.e_1 \mid \rho_1 \preccurlyeq \forall x_2 : a_2.a_4\ \rho_2$. Similar to previous case.

- Case $\phi;\theta \vdash w \preccurlyeq \lambda x_2 : a_2.e_2\rho_2$. We prove the case where the first premise is $w{\uparrow} \xrightarrow[\theta]{\mu}{\triangleright}^* \lambda x_1 : a_1.e_1 \mid \rho_1$; the other case is similar. We need to show $w\delta = \lambda x_2 : a_2.e_2\rho_2\delta$. By assumption, we have (1) $\vdash_\delta w$, (2) $\vdash_\delta \lambda x_2 : a_2.e_2\rho_2$ and (3) $w \mid \delta = \lambda x_2 : a_2.e_2 \mid \rho_2\delta$. By (1), Proposition 5 applies to $w{\uparrow} \xrightarrow[\theta]{\mu}{\triangleright}^* \lambda x_1 : a_1.e_1 \mid \rho_1$, giving (4) $\vdash_\delta \lambda x_1 : a_1.e_1 \mid \rho_1$ and (5) $w\delta = \lambda x_1 : a_1.e_1 \mid \rho_1\delta$. Let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. Freshness of $\mathrm{v}_k^{a_i\rho_i}$ implies $\vdash_\delta \rho_i \iff \vdash_{\delta'} \rho_i$ and $\rho_i\delta = \rho_i\delta'$. In order to apply the induction hypothesis to $\phi;\theta \vdash w\ \mathrm{v}_k^{a_1\rho_1} \preccurlyeq e_2\rho_2'$, we first need to show (6) $\vdash_{\delta'} w\ \mathrm{v}_k^{a_1\rho_1}$, (7) $\vdash_{\delta'} e_2\rho_2'$ and (8) $(w\ \mathrm{v}_k^{a_1\rho_1}) \mid \delta' = e_2\rho_2' \mid \delta'$. Condition (6) follows from (1), (4) and (5). Condition (7) follows from (2) as in the previous case. We also have

$$
\begin{aligned}
& (w\ \mathrm{v}_k^{a_1\rho_1}) \mid \delta' \\
= \quad & \{\text{semantics of values}\} \\
& (w \mid \delta') * (\mathrm{v}_k^{a_1\rho_1}\delta') \\
\sqsubseteq \quad & \{(3), \text{ and } a_1\rho_1\delta' \sqsubseteq a_2\rho_2\delta' \text{ by (3), (5) and injectivity of } \mathsf{Prod}\ \} \\
& (\lambda x_2 : a_2.e_2\rho_2 \mid \delta') * (\mathrm{v}_k^{a_2\rho_2}\delta') \\
= \quad & \{\text{definition of semantic type application}\} \\
& e_2 \mid (\rho_2\delta', x_2 : \mathsf{p}_{\mathsf{V}}\, a_2\rho_2\delta' = d) \\
= \quad & \{\text{semantics of values}\} \\
& e_2\rho_2' \mid \delta'
\end{aligned}
$$

so that induction hypothesis now applies to $\phi;\theta \vdash w\ \mathrm{v}_k^{a_1\rho_1} \preccurlyeq e_2\rho_2'$, giving $(w\ \mathrm{v}_k^{a_1\rho_1})\delta' \sqsubseteq e_2\rho_2'\delta'$. Thus for all $d \in \mathsf{D}$ we have

$$
\begin{aligned}
& (w\delta)\ (\mathsf{p}_{a_1\rho_1\delta}\, d) \\
= \quad & \{\text{semantics of values, semantic function application, } \vdash_\delta a_1\rho_1 : \mathsf{V}\} \\
& (w\ \mathrm{v}_k^{a_1\rho_1})\delta' \\
\sqsubseteq \quad & \{\text{induction hypothesis (see above)}\} \\
& e_2\rho_2'\delta' \\
= \quad & \{\text{semantics of expressions and values}\} \\
& (\lambda x_2 : a_2.e_2\rho_2\delta)\ (\mathsf{p}_{a_2\rho_2\delta}\, d)
\end{aligned}
$$

and by Lemma 10 (since $w \mid \delta \sqsubseteq \lambda x_2 : a_2.e_2\rho_2 \mid \delta$) we obtain $w\delta \sqsubseteq \lambda x_2 : a_2.e_2\rho_2\delta$, as required.

- Case $\phi;\theta \vdash \lambda x_1 : a_1.e_1\rho_1 \preccurlyeq w$. Similar to previous case.

- Case $\phi;\theta \vdash \langle\bullet\rangle\rho_1 \preccurlyeq \langle\bullet\rangle\rho_2$. Immediate.

- Case $\phi;\theta \vdash \langle x : a_1, s_1\rangle\rho_1 \preccurlyeq \langle x : a_2, s_2\rangle\rho_2$. We need to show $\langle x : a_1, s_1\rangle\rho_1\delta \sqsubseteq \langle x : a_2, s_2\rangle\rho_2\delta$, that is (1) $\mathsf{p_V}\, a_1\rho_1\delta \sqsubseteq \mathsf{p_V}\, a_2\rho_2\delta$ and (2) $\lambda d.\langle s_1\rangle(\rho_1\delta, x : \mathsf{p_V}\, a_1\rho_1\delta = d) \sqsubseteq \lambda d.\langle s_2\rangle(\rho_2\delta, x : \mathsf{p_V}\, a_2\rho_2\delta = d)$. By inversion of $\vdash_\delta \langle x : a_i, s_i\rangle\rho_i$, we have (3) $\vdash_\delta \rho_i$, (4) $\rho_i\delta \vdash a_i : \mathsf{V}$ and (5) $\forall d.\ (\rho_i\delta, x : \mathsf{p_V}\, a_i\rho_i\delta = d) \vdash \langle s_i\rangle$. Thus $\vdash_\delta a_i\rho_i$ by (3) and (4), and $a_1\rho_1 \mid \delta \sqsubseteq a_2\rho_2 \mid \delta$ by (4). Now induction hypothesis applies to $\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2$, giving $\mathsf{p_V}\, a_1\rho_1\delta = a_1\rho_1\delta \sqsubseteq a_2\rho_2\delta = \mathsf{p_V}\, a_2\rho_2\delta$ by (4), as required for (1). Let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. We have $\vdash_{\delta'} \langle s_i\rangle(\rho_i, x = \mathrm{v}_k^{a_i\rho_i})$ by (3–5) and by freshness of $\mathrm{v}_k^{a_i\rho_i}$. Hence induction hypothesis applies to $\phi;\theta \vdash \langle s_1\rangle(\rho_1, x = \mathrm{v}_k^{a_1\rho_1}) \preccurlyeq \langle s_2\rangle(\rho_2, x = \mathrm{v}_k^{a_2\rho_2})$, giving $\forall d.\ \langle s_1\rangle(\rho_1\delta, x : \mathsf{p_V}\, a_1\rho_1\delta = d) \sqsubseteq \langle s_2\rangle(\rho_2\delta, x : \mathsf{p_V}\, a_2\rho_2\delta = d)$, as required for (2).

- Case $\phi;\theta \vdash \langle x = e_1, s_1\rangle\rho_1 \preccurlyeq \langle x = e_2, s_2\rangle\rho_2$. Similar to previous case.

- Case $\phi;\theta \vdash \langle r_1 :: a_1\rangle\rho_1 \preccurlyeq \langle r_2 :: a_2\rangle\rho_2$. We need to show $\langle r_1 :: a_1\rangle\rho_1\delta = \tau\ (\mathsf{p_V}\, a_1\rho_1\delta)\ (r_1\rho_1\delta) \sqsubseteq \tau\ (\mathsf{p_V}\, a_2\rho_2\delta)\ (r_2\rho_2\delta) = \langle r_2 :: a_2\rangle\rho_2\delta$. It thus suffices to show (1) $\mathsf{p_V}\, a_1\rho_1\delta \sqsubseteq \mathsf{p_V}\, a_2\rho_2\delta$ and (2) $r_1\rho_1\delta \sqsubseteq r_2\rho_2\delta$. Condition (1) holds by assumption $\langle r_1 :: a_1\rangle\rho_1 \mid \delta \sqsubseteq \langle r_2 :: a_2\rangle\rho_2 \mid \delta$. By inversion of $\vdash_\delta \langle r_i :: a_i\rangle\rho_i$, we have (3) $\vdash_\delta \rho_i$, (4) $\rho_i\delta \vdash a_i : \mathsf{V}$, (5) $a_i\rho_i\delta = [xs_i : s_i]$ and (6) $\rho_i\delta \models r_i : [xs_i : s_i]$. Conditions (1), (4) and (5) imply that (7) $[xs_1 : s_1] \sqsubseteq [xs_2 : s_2]$. By assumption, we have (8) $\vdash_\delta \phi$ and (9) $\vdash_\delta \theta$. Inversion of $\phi;\theta \vdash \langle r_1 :: a_1\rangle\rho_1 \preccurlyeq \langle r_2 :: a_2\rangle\rho_2$ gives (10) $\phi;\theta \models r_1\rho_1 \preccurlyeq r_2\rho_2$. We now show that (3) and (6–10) imply (2), by induction on $xs_1 = xs_2$.

  - Case $\langle\rangle$. Since $\rho_i\delta \models r_i : []$, we must have $r_i = ()$. Thus (2) is immediate.
  - Case $\langle x^-, xs\rangle$. Then $[xs_i : s_i] = [x : u_i, xs : f_i]$. By (6), we have $\rho_i\delta \models r_i : [x : u_i, xs : f_i]$, where r_i must be of the form $(x = e_i, r'_i)$. By inversion of (6), we thus obtain (11) $\rho_i\delta \vdash e_i : u_i$ and (12) $\rho_i\delta \models r'_i : [xs : f_i(e_i\rho_i\delta)]$. Furthermore, inversion of (10) gives (13) $\phi;\theta \vdash e_1\rho_1 \preccurlyeq e_2\rho_2$ and (14) $\phi;\theta \models r'_1\rho_1 \preccurlyeq r'_2\rho_2$. Now (7) and (11) imply that (15) $e_1 \mid \rho_1\delta \sqsubseteq e_2 \mid \rho_2\delta$. By (3), (8), (9), (11) and (15), induction hypothesis applies to (13), giving (16) $e_1\rho_1\delta \sqsubseteq e_2\rho_2\delta$. By (7) and (16), we have (17) $f_1(e_1\rho_1\delta) \sqsubseteq f_2(e_2\rho_2\delta)$. By (3), (8), (9), (12) and (17), induction hypothesis applies to (14), giving (18) $r'_1\rho_1\delta \sqsubseteq r'_2\rho_2\delta$. Now conditions (16) and (18) together give (2), as required.
  - Case $\langle x^+, xs\rangle$. By (6), we have $\rho_i\delta \models r_i : [x : u_i = d_i, xs : s'_i]$, and by inversion we obtain $\rho_i\delta \models r_i : [xs : s'_i]$. By (7), we have $s'_1 \sqsubseteq s'_2$. The desired result now follows immediately by induction hypothesis.

- Case $\phi;\theta \vdash \mathrm{v}_k^{t_1} \preccurlyeq \mathrm{v}_k^{t_2}$. By inversion of $\mathrm{v}_k^{t_i}$, we have $\vdash_\delta t_i : \mathsf{V}$. Now induction hypothesis applies to $\phi;\theta \vdash t_1 \preccurlyeq t_2$, giving $t_1\delta \sqsubseteq t_2\delta$. Thus for $\delta(k) = d$, we have $\mathrm{v}_k^{t_1} \mid \delta = \mathsf{p_V}\, t_1\delta \sqsubseteq \mathsf{p_V}\, t_2\delta = \mathrm{v}_k^{t_2} \mid \delta$ and $\mathrm{v}_k^{t_1}\delta = \mathsf{p}_{\mathsf{p_V}\, t_1\delta}\, d \sqsubseteq \mathsf{p}_{\mathsf{p_V}\, t_2\delta}\, d = \mathrm{v}_k^{t_2}\delta$, as required.

- Case $\phi;\theta \vdash w_1\ v_1 \preccurlyeq w_2\ v_2$. By inversion of $\vdash_\delta w_i\ v_i$, we have (1) $\vdash_\delta w_i$, (2) $w_i \mid \delta = \mathsf{Prod}\ (v_i \mid \delta)\ f_i$ and (3) $\vdash_\delta v_i$. Induction hypothesis on $\phi;\theta \vdash w_1 \preccurlyeq w_2$ gives (4) $w_1\delta \sqsubseteq w_2\delta$ and (5) $w_1 \mid \delta \sqsubseteq w_2 \mid \delta$. By (2), (5) and injectivity of Prod , we then have (6) $v_1 \mid \delta \sqsubseteq v_2 \mid \delta$. By (3) and (6), induction hypothesis applies to $\phi;\theta \vdash v_1 \preccurlyeq v_2$, giving $v_1\delta \sqsubseteq v_2\delta$. Hence by (4–6) we have $(w_1\ v_1)\delta = w_1\delta \cdot v_1\delta \sqsubseteq w_2\delta \cdot v_2\delta = (w_2\ v_2)\delta$ and $(w_1\ v_1) \mid \delta = (w_1 \mid \delta) * v_1\delta \sqsubseteq (w_2 \mid \delta) * v_2\delta = (w_2\ v_2) \mid \delta$, as required.

- Case $\phi;\theta \vdash w_1 * v_1 \preccurlyeq w_1 * v_2$. Similar to previous case.
- Case $\phi;\theta \vdash \pi_c\ w_1 \preccurlyeq \pi_c\ w_2$. The desired conlusions follow from inversion of $\vdash_\delta \pi_c\ w_i$ and induction hypothesis on $\phi;\theta \vdash w_1 \preccurlyeq w_2$.
- Case $\phi;\theta \vdash \mu x_1 : a_1.e_1\rho_1 \preccurlyeq \mu x_2 : a_2.e_2\rho_2$. Inversion of $\vdash_\delta \mu x_i : a_i.e_i\rho_i$ gives (1) $\vdash_\delta \rho_i$, (2) $\rho_i\delta \vdash a_i : \mathsf{V}$ and (3) $\forall d.\ (\rho_i\delta, x_i : \mathsf{p_V}\, a_i\rho_i\delta = d) \vdash e_i : a_i\rho_i\delta$. Thus $\vdash_\delta a_i\rho_i : \mathsf{V}$ by (1) and (2), so that induction hypothesis applies to $\phi;\theta \vdash a_1\rho_1 \preccurlyeq a_2\rho_2$, giving (4) $\mathsf{p_V}\, a_1\rho_1\delta = a_1\rho_1\delta \sqsubseteq a_2\rho_2\delta = \mathsf{p_V}\, a_2\rho_2\delta$. By (3) and (4), we have $\forall d.\ e_1 \mid (\rho_1\delta, x_1 : \mathsf{p_V}\, a_1\rho_1\delta = d) \sqsubseteq e_2 \mid (\rho_2\delta, x_2 : \mathsf{p_V}\, a_2\rho_2\delta = d)$. We need to show $\mu x_1 : a_1.e_1\rho_1\delta \sqsubseteq \mu x_2 : a_2.e_2\rho_2\delta$, where $\mu x_i : a_i.e_i\rho_i\delta = \mathsf{fix}\,(\lambda d.\mathsf{p}_{\mathsf{p_V}\, a_i\rho_i\delta}\, e_i(\rho_i\delta, x_i : \mathsf{p_V}\, a_i\rho_i\delta = d))$. By (2) and (3), it suffices to show $\forall d.\ e_1(\rho_1\delta, x_1 : \mathsf{p_V}\, a_1\rho_1\delta = d) \sqsubseteq e_2(\rho_2\delta, x_2 : \mathsf{p_V}\, a_2\rho_2\delta = d)$. Let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. Freshness of $\mathrm{v}_k^{a_i\rho_i}$ implies that $\vdash_\delta \rho_i \iff \vdash_{\delta'} \rho_i$ and $\rho_i\delta = \rho_i\delta'$. Hence we have $\vdash_{\delta'} e_i(\rho_i, x_i = \mathrm{v}_k^{a_i\rho_i})$ by (1–3) and freshness of $\mathrm{v}_k^{a_i\rho_i}$, and we have $e_1\rho_1' \mid \delta' \sqsubseteq e_2\rho_2' \mid \delta'$ by (4) and freshness of $\mathrm{v}_k^{a_i\rho_i}$. Now induction hypothesis applies to $\phi;\theta \vdash e_1\rho_1' \preccurlyeq e_2\rho_2'$, giving $e_1(\rho_1\delta, x_1 : u = d) = e_1\rho_1'\delta' \sqsubseteq e_2\rho_2'\delta' = e_2(\rho_2\delta, x_2 : u = d)$ as desired.
- Case $\phi;\theta \vdash \sigma_x\ v_1\ t_1 \preccurlyeq \sigma_x\ v_2\ t_2$. We need to show that $(\sigma_x\ v_1\ t_1)\delta \sqsubseteq (\sigma_x\ v_2\ t_2)\delta$ and $(\sigma_x\ v_1\ t_1) \mid \delta \sqsubseteq (\sigma_x\ v_2\ t_2) \mid \delta$. Inversion of $\vdash_\delta \sigma_x\ v_i\ t_i$ gives (1) $\vdash_\delta t_i$, (2) $\vdash_\delta v_i$ and (3) $v_i \mid \delta = t_i\delta = [xs_i : s_i]$ where $x \in xs_i$. By (1) and (3), induction hypothesis applies to $\phi;\theta \vdash t_1 \preccurlyeq t_2$ giving (4) $t_1\delta \sqsubseteq t_2\delta$. By (2) and (3), we thus have $\vdash_\delta v_i$ and $v_1 \mid \delta \sqsubseteq v_2 \mid \delta$, so that induction hypothesis now applies to $\phi;\theta \vdash v_1 \preccurlyeq v_2$, giving (5) $v_1\delta \sqsubseteq v_2\delta$. Hence by (4) and (5) we have $(\sigma_x\ v_1\ t_1)\delta = \sigma_x\ (v_1\delta)\ (t_1\delta) \sqsubseteq \sigma_x\ (v_1\delta)\ (t_1\delta) = (\sigma_x\ v_2\ t_2)\delta$ and $(\sigma_x\ v_1\ t_1) \mid \delta = \Sigma_x\ (v_1\delta)\ (t_1\delta) \sqsubseteq \Sigma_x\ (v_1\delta)\ (t_1\delta) = (\sigma_x\ v_2\ t_2) \mid \delta$, as required.
- Case $\phi;\theta \vdash \Sigma_x\ v_1\ t_1 \preccurlyeq \Sigma_x\ v_2\ t_2$. Similar to the previous case.
- Case $\phi;\theta \vdash (c\ e_1\ a_1)\rho_1 \preccurlyeq (c\ e_2\ a_2)\rho_2$. Inversion of $\vdash_\delta (c\ e_i\ a_i)\rho_i$ gives (1) $\vdash_\delta \rho_i$, (2) $\rho_i\delta \vdash a_i : \mathsf{V}$, (3) $a_i\rho_i\delta = \mathsf{Data}\ \Delta_i$, (4) $c\Delta_i = \mathsf{Def}\ u_i$ and (5) $\rho_i\delta \vdash e_i : u_i$. We need to show $(c\ e_1\ a_1)\rho_1\delta = \mathsf{p}_{\mathsf{p_V}\, a_1\rho_1\delta}(\mathsf{Inj}_c\ (e_1\rho_1\delta)) \sqsubseteq \mathsf{p}_{\mathsf{p_V}\, a_2\rho_2\delta}(\mathsf{Inj}_c\ (e_2\rho_2\delta)) = (c\ e_2\ a_2)\rho_2\delta$. Assumption $(c\ e_1\ a_1)\rho_1 \mid \delta \sqsubseteq (c\ e_2\ a_2)\rho_2 \mid \delta$ gives (6) $a_1\rho_1\delta = \mathsf{p_V}\, a_1\rho_1\delta \sqsubseteq \mathsf{p_V}\, a_2\rho_2\delta = a_2\rho_2\delta$, by (1) and (2). By monotonicity of $\mathsf{p}_{(-)}$, it thus suffices to show (7) $e_1\rho_1\delta \sqsubseteq e_2\rho_2\delta$. By (3), (6) and injectivity of Data , we have $\Delta_1 \sqsubseteq \Delta_2$. By (4) and (5), we then obtain $e_1\rho_1 \mid \delta \sqsubseteq e_2\rho_2 \mid \delta$. Since we also have $\vdash_\delta e_i\rho_i$ by (1) and (5), induction hypothesis now applies to $\phi;\theta \vdash e_1\rho_1 \preccurlyeq e_2\rho_2$, giving (7) as required.
- Case $\phi;\theta \vdash \{d_1\}\rho_1 \preccurlyeq \{d_2\}\rho_2$. We need to show $\{d_1\}\rho_1\delta = \mathsf{Data}\ \Delta_1 \sqsubseteq \mathsf{Data}\ \Delta_2 = \{d_2\}\rho_2\delta$, where $\Delta_i = \mathit{data}\ d_i\ (\rho_i\delta)$. Since $c\Delta_i = \bot$ for any $c \notin \mathit{dom}(d_1)$, it suffices to show $\forall c \in \mathit{dom}(d_1).\ c\Delta_1 \sqsubseteq c\Delta_2$, that is (1) $\forall c \in \mathit{dom}(d_1).\ c \in \mathit{dom}(d_2)$ and (2) $\forall c \in \mathit{dom}(d_1).\ \pi_c(\{d_1\}\rho_1\delta) \sqsubseteq \pi_c(\{d_2\}\rho_2\delta)$. Condition (1) is immediate from the first premise. Condition (2) follows from (1) and induction hypothesis on the premise $\phi;\theta \vdash \pi_c(\{d_1\}\rho_1) \preccurlyeq \pi_c(\{d_2\}\rho_2)$.

- $\phi;\theta \vdash w \preccurlyeq \langle r :: a\rangle\rho$. Inversion of $\vdash_\delta \langle r :: a\rangle\rho$ gives (1) $\vdash_\delta \rho$, (2) $\rho\delta \vdash a : \mathsf{V}$ and (3) $a\rho\delta = [xs : s']$. Proposition 5 applied to the premise $w\!\uparrow \xrightarrow[\theta]{\mu}{}^{*} \langle s_w\rangle\rho'$ gives (4) $w \mid \delta = [xs : s]$, where $xs = dom(s_w)$. By assumption and (1–4), we thus have (5) $w \mid \delta = [xs : s] \sqsubseteq [xs : s'] = a\rho\delta$. For $[xs : s]$, we define

$$[xs : s]_0 = [xs : s]$$

$$[xs : s]_{i+1} = \begin{cases} [xs_{i+1} : f_{i+1}(\mathsf{p}_{u_i}(x_i(w\delta)))] & \text{if } [xs : s]_i = [x_i : u_i, xs_{i+1} : f_{i+1}] \\ [xs_{i+1} : s_{i+1}] & \text{if } [xs : s]_i = [x_i : u_i = d_i, xs_{i+1} : s_{i+1}] \\ [] & \text{if } [xs : s]_i = [] \end{cases}$$

and similarly for $[xs : s']$. We need to show $w\delta \sqsubseteq \langle r :: a\rangle\rho\delta$. It suffices to show $\forall i \in [0..l).\ x_i(w\delta) \sqsubseteq x_i(\langle r :: a\rangle\rho\delta)$, where l is the length of xs, since both sides equal $\bot$ for any x_i not occuring in xs. To prove this, attempting an induction on xs fails at the inductive step because the property $\vdash_\delta w : [xs : s]$ cannot be maintained. We therefore generalise $w\delta \sqsubseteq \langle r :: a\rangle\rho\delta$ to the following statement.

$$\begin{array}{l} \forall n. \\ \quad (\forall i \in [0..min(n,l)).\ x_i(w\delta) \sqsubseteq x_i(\langle r :: a\rangle\rho\delta)) \wedge \\ \quad [xs : s]_{min(n,l)} \sqsubseteq [xs : s']_{min(n,l)} \wedge \\ \quad \forall i \in [min(n,l)..l). \\ \qquad \left(\begin{array}{c} \sigma_{x_i}\ (w\delta)\ (w \mid \delta) = \sigma_{x_i}\ (w\delta)\ [xs : s]_{min(n,l)} \wedge \\ \Sigma_{x_i}\ (w\delta)\ (w \mid \delta) = \Sigma_{x_i}\ (w\delta)\ [xs : s]_{min(n,l)} \wedge \\ \sigma_{x_i}\ (\langle r :: a\rangle\rho\delta)\ (a\rho\delta) = \sigma_{x_i}\ (\langle r :: a\rangle\rho\delta)\ [xs : s']_{min(n,l)} \wedge \\ \Sigma_{x_i}\ (\langle r :: a\rangle\rho\delta)\ (a\rho\delta) = \Sigma_{x_i}\ (\langle r :: a\rangle\rho\delta)\ [xs : s']_{min(n,l)} \wedge \\ x_i(\mathsf{p}_{[xs:s]_{min(n,l)}}(w\delta)) = x_i(w\delta) \wedge \\ x_i(\mathsf{p}_{[xs:s']_{min(n,l)}}(\langle r :: a\rangle\rho\delta)) = x_i(\langle r :: a\rangle\rho\delta) \end{array} \right) \end{array}$$

We now prove the above by induction on n.

  - Case $n = 0$. Follows from assumptions (5), $\vdash_\delta w : [xs : s]$ and $\vdash_\delta \langle r :: a\rangle\rho : [xs : s']$.
  - Case $n = k + 1$, where $k + 1 < l$. Note that $min(k + 1, l) = k + 1$. First we show that $\forall i \in [0..k + 1).\ x_i(w\delta) \sqsubseteq x_i(\langle r :: a\rangle\rho\delta)$; by induction hypothesis it suffices to show $x_k(w\delta) \sqsubseteq x_k(\langle r :: a\rangle\rho\delta)$. Since $\vdash_\delta \sigma_{x_k}\ w\ (w\!\uparrow)$ and $\vdash_\delta \sigma_{x_k}\ (\langle r :: a\rangle\rho)\ (a\rho)$ follow from $k + 1 < l$, (5) and assumptions, and since we have

$$
\begin{aligned}
& \Sigma_{x_k}\ (w\delta)\ (w \mid \delta) \\
= & \qquad\qquad \{\text{induction hypothesis}\} \\
& \Sigma_{x_k}\ (w\delta)\ [xs : s]_k \\
= & \qquad\qquad \{\text{definition of } \Sigma_{x_k}\ (-)\ (-)\} \\
& u_k \\
\sqsubseteq & \qquad\qquad \{\text{since } [xs : s]_k \sqsubseteq [xs : s']_k \text{ by induction hypothesis}\} \\
& u'_k \\
= & \qquad\qquad \{\text{definition of } \Sigma_{x_k}\ (-)\ (-)\} \\
& \Sigma_{x_k}\ (\langle r :: a\rangle\rho\delta)\ [xs : s']_k \\
= & \qquad\qquad \{\text{induction hypothesis}\} \\
& \Sigma_{x_k}\ (\langle r :: a\rangle\rho\delta)\ (a\rho\delta)
\end{aligned}
$$

the induction hypothesis applies to $\phi; \theta \vdash \sigma_{x_k}\ w\ (w\uparrow) \preccurlyeq \sigma_{x_k}\ (\langle r :: a\rangle\rho)\ (a\rho)$, giving $\sigma_{x_k}\ w\delta\ (w \mid \delta) \sqsubseteq \sigma_{x_k}\ (\langle r :: a\rangle\rho\delta)\ (a\rho\delta)$. For $x_k(w\delta) \sqsubseteq x_k(\langle r :: a\rangle\rho\delta)$, it thus remains to show that $x_k(w\delta) = \sigma_{x_k}\ w\delta\ (w \mid \delta)$ and $x_k(\langle r :: a\rangle\rho\delta) = \sigma_{x_k}\ (\langle r :: a\rangle\rho\delta)\ (a\rho\delta)$. We prove $x_k(w\delta) = \sigma_{x_k}\ w\delta\ (w \mid \delta)$ for the subcase where $[xs : s]_k = [x_k : u_k = d_k, xs_{k+1} : s_{k+1}]$; the subcase where $[xs : s]_k = [x_k : u_k, xs_{k+1} : f_{k+1}]$ and the proof for $x_k(\langle r :: a\rangle\rho\delta) = \sigma_{x_k}\ (\langle r :: a\rangle\rho\delta)\ (a\rho\delta)$ are similar.

$$
\begin{aligned}
& x_k(w\delta) \\
= & \qquad\qquad \{\text{induction hypothesis}\} \\
& x_k(\mathsf{p}_{[xs:s]_k}(w\delta)) \\
= & \qquad\qquad \{\text{subcase assumption}\} \\
& x_k(\mathsf{p}_{[x_k:u_k=d_k,xs_{k+1}:s_{k+1}]}(w\delta)) \\
= & \qquad\qquad \{\text{definition of } \mathsf{p}_{(-)}\} \\
& x_k(\mathsf{p}_{[xs_{k+1}:s_{k+1}]}(w\delta), x_k = \mathsf{p}_{u_k}(d_k \sqcap x_k(w\delta))) \\
= & \qquad\qquad \{\text{semantic record field lookup}\} \\
& \mathsf{p}_{u_k}(d_k \sqcap x_k(w\delta))
\end{aligned}
$$

By applying p_{u_k} to both sides of the above equation, and by idempotency of finitary projections, we obtain $\mathsf{p}_{u_k} x_k(w\delta) = \mathsf{p}_{u_k}(d_k \sqcap x_k(w\delta))$. Combining these two equations gives $x_k(w\delta) = \mathsf{p}_{u_k} x_k(w\delta)$. Thus we have

$$
\begin{array}{ll}
& \sigma_{x_k}\ w\delta\ (w \mid \delta) \\
= & \hfill \{\text{induction hypothesis}\} \\
& \sigma_{x_k}\ w\delta\ [xs : s]_k \\
= & \hfill \{\text{definition of } \sigma_{x_k}\ (-)\ (-)\} \\
& \mathsf{p}_{u_k} x_k(w\delta) \\
= & \hfill \{x_k(w\delta) = \mathsf{p}_{u_k} x_k(w\delta)\} \\
& x_k(w\delta)
\end{array}
$$

as required for $x_k(w\delta) \sqsubseteq x_k(\langle r :: a\rangle\rho\delta)$.
Next, note that $[xs : s]_{k+1} \sqsubseteq [xs : s']_{k+1}$ follows from induction hypothesis and from the proof of $x_k(w\delta) \sqsubseteq x_k(\langle r :: a\rangle\rho\delta)$. Also note that $\forall i \in [k+1..l).\ \sigma_{x_i}\ (w\delta)\ (w \mid \delta) = \sigma_{x_i}\ (w\delta)\ [xs : s]_{k+1}$ and the three other similar equations follow from induction hypothesis $\forall i \in [k..l).\ \sigma_{x_i}\ (w\delta)\ (w \mid \delta) = \sigma_{x_i}\ (w\delta)\ [xs : s]_k$ and from the definitions of $\sigma_{x_i}\ (-)\ (-)$ and $\Sigma_{x_i}\ (-)\ (-)$ for $i \neq k$.
Finally, we show that $\forall i \in [k+1..l).\ x_i(\mathsf{p}_{[xs:s]_{k+1}}(w\delta)) = x_i(w\delta)$ for the subcase where $[xs : s]_k = [x_k : u_k = d_k, xs_{k+1} : s_{k+1}]$; the subcase for $[xs : s]_k = [x_k : u_k, xs_{k+1} : f_{k+1}]$ and the proof for $\forall i \in [k+1..l).\ x_i(\mathsf{p}_{[xs:s']_{k+1}}(\langle r :: a\rangle\rho\delta)) = x_i(\langle r :: a\rangle\rho\delta)$ are similar. Let $i \in [k+1..l)$. Then we have

$$
\begin{array}{ll}
& x_i(w\delta) \\
= & \hfill \{\text{induction hypothesis}\} \\
& x_i(\mathsf{p}_{[xs:s]_k}(w\delta)) \\
= & \hfill \{\text{subcase assumption}\} \\
& x_i(\mathsf{p}_{[x_k:u_k=d_k, xs_{k+1}:s_{k+1}]}(w\delta)) \\
= & \hfill \{\text{definition of } \mathsf{p}_{(-)}\} \\
& x_i(\mathsf{p}_{[xs_{k+1}:s_{k+1}]}(w\delta), x_k = \mathsf{p}_{u_k}(d_k \sqcap x_k(w\delta))) \\
= & \hfill \{x_i \neq x_k \text{ for } i > k, \text{ since record field names are distinct}\} \\
& x_i(\mathsf{p}_{[xs_{k+1}:s_{k+1}]}(w\delta))
\end{array}
$$

as required.

- Case $n = k+1$, where $k+1 = l$. Similar to previous case.
- Case $n = k+1$, where $k+1 > l$. Immediate from induction hypothesis.

- $\phi; \theta \vdash \langle r :: a\rangle\rho \preccurlyeq w$. Similar to previous case.
- Case $\phi; \theta \vdash h \preccurlyeq g$. We need to show that $h\delta \sqsubseteq g\delta$. By induction on g, we have that (1) $\vdash_\delta w = \textsc{scrutinee}(g)$. Thus Proposition 5 applies to $w{\uparrow} \overset{\mu}{\underset{\theta}{\rhd}}{}^{*} \{d\}\rho'$, giving (2)

$\vdash_\delta \{d\}\rho'$ and (3) $w \mid \delta = \{d\}\rho'\delta = \mathsf{Data}\ \Delta$. Since $w\delta : \mathsf{Data}\ \Delta$, we must have either $w\delta = \bot$ or $w\delta = \mathsf{Inj}_c\ d'$, for some $c \in dom(d)$ and $d' \in \mathsf{D}$.

- Case $w\delta = \bot$. Then by an induction on g, we also have that $g\delta = \bot$, so it suffices to show $h\delta \sqsubseteq \bot$. By assumption $\vdash_\delta g$, we have $\vdash_\delta \perp\!\!\!\perp^{(g\uparrow)}$, and thus by assumption $h \mid \delta \sqsubseteq g \mid \delta$, we have $h \mid \delta \sqsubseteq \perp\!\!\!\perp^{(g\uparrow)} \mid \delta$. Similarly, by $\vdash_\delta w$ we have $\vdash_\delta \perp\!\!\!\perp^{(w\uparrow)}$, as well as $w\delta = \bot = \perp\!\!\!\perp^{(w\uparrow)}\delta$ and $w \mid \delta = \perp\!\!\!\perp^{(w\uparrow)} \mid \delta$. Thus induction hypothesis applies to $\phi; (\theta, w \approx \perp\!\!\!\perp^{(w\uparrow)}) \vdash h \preccurlyeq \perp\!\!\!\perp^{(g\uparrow)}$, giving $h\delta \sqsubseteq \perp\!\!\!\perp^{(g\uparrow)}\delta = \bot$, as desired.
- Case $w\delta = \mathsf{Inj}_c\ d'$, for some $c \in dom(d)$ and $d' \in \mathsf{D}$. Consider $\delta' = (\delta, k = d')$. Freshness of $\mathrm{v}_k^{\pi_c(\{d\}\rho')}$ implies that $\vdash_\delta \theta \iff \vdash_{\delta'} \theta$, $\vdash_\delta \rho' \iff \vdash_{\delta'} \rho'$ and $\rho'\delta = \rho'\delta'$. To show that $\vdash_{\delta'} \theta_c$, we need to show (4) $\vdash_{\delta'} (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho')$, (5) $w\delta' = (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho')\delta'$ and (6) $w \mid \delta' = (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho') \mid \delta'$. Condition (4) follows from (2), (3), assumption $c \in dom(d)$ and freshness of $\mathrm{v}_k^{\pi_c(\{d\}\rho')}$. For (5) and (6) we have

$$
\begin{aligned}
& (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho')\delta' \\
= \quad & \{\text{semantics of expressions}\} \\
& \mathsf{p}_{\mathsf{pV}(y(x=\mathrm{v}_k^{\pi_c(\{d\}\rho')}, y=\{d\}\rho')\delta')}(\mathsf{Inj}_c\ (x(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho')\delta')) \\
= \quad & \{(3) \text{ and freshness of } \mathrm{v}_k^{\pi_c(\{d\}\rho')}\} \\
& \mathsf{p}_{(\mathsf{Data}\ \Delta)}(\mathsf{Inj}_c\ (\mathsf{p}_{\mathsf{pV}(\pi_c(\{d\}\rho')\delta')}\, d')) \\
= \quad & \{c\Delta = \mathsf{Def}\ u\} \\
& \mathsf{p}_{(\mathsf{Data}\ \Delta)}(\mathsf{Inj}_c\ (\mathsf{p}_u\, d')) \\
= \quad & \{c\Delta = \mathsf{Def}\ u\} \\
& \mathsf{p}_{(\mathsf{Data}\ \Delta)}(\mathsf{p}_{(\mathsf{Data}\ \Delta)}(\mathsf{Inj}_c\ d')) \\
= \quad & \{\mathsf{Inj}_c\ d' = w\delta : \mathsf{Data}\ \Delta\} \\
& w\delta
\end{aligned}
$$

and

$$
\begin{aligned}
& (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho') \mid \delta' \\
= \quad & \{\text{semantics of expressions}\} \\
& \mathsf{pV}(\{d\}\rho'\delta') \\
= \quad & \{\text{freshness of } \mathrm{v}_k^{\pi_c(\{d\}\rho')}\} \\
& \mathsf{pV}(\{d\}\rho'\delta) \\
= \quad & \{(3)\} \\
& w \mid \delta
\end{aligned}
$$

respectively. We have thus shown that $\vdash_{\delta'} \theta_c$, for $c \in dom(d)$. Hence induction hypothesis applies to $\phi; \theta_c \vdash h \preccurlyeq g$, giving $h\delta \sqsubseteq g\delta$, as required.

- Case $\phi;\theta \vdash h \preccurlyeq g$. Similar to previous case.
- Case $\phi;\theta \vdash \underline{w}_1 \preccurlyeq h_1$, where $(\underline{w}_2 \preccurlyeq h_2) \in \phi$, $\underline{w}_1 \approx \underline{w}_2$, $h_1{\uparrow} \approx h_2{\uparrow}$, $h_i \xrightarrow[\theta]{\mu}{}^{*} h_i'$ and $h_1' \approx h_2'$. Since $(\underline{w}_2 \preccurlyeq h_2) \in \phi$, our assumptions include $\underline{w}_2\delta \sqsubseteq h_2\delta$ and $\underline{w}_2 \mid \delta \sqsubseteq h_2 \mid \delta$. The desired result thus follows from Propositions 4 and 5.
- Case $\phi;\theta \vdash w \preccurlyeq h$, where $(\phi, \mathrm{G}(\mathrm{v}_k^{a\rho}) \preccurlyeq h);\theta \vdash \mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho})) \preccurlyeq h$ and $\phi;\theta \vdash \mathrm{G}(\mathrm{v}_k^{a\rho}){\uparrow} \preccurlyeq h{\uparrow}$. By assumption, w must be contained in the grammar $\overline{w} := \mu x : a.e\rho \mid \overline{w}\ v$.

  Let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. By freshness of $\mathrm{v}_k^{a\rho}$, we have $\vdash_\delta \phi \iff \vdash_{\delta'} \phi$, $\vdash_\delta \theta \iff \vdash_{\delta'} \theta$, $\phi\delta = \phi\delta'$, $\vdash_\delta w \iff \vdash_{\delta'} w$ and $\vdash_\delta h \iff \vdash_{\delta'} h$. For applying induction hypothesis to $\phi;\theta \vdash \mathrm{G}(\mathrm{v}_k^{a\rho}){\uparrow} \preccurlyeq h{\uparrow}$, we need to establish $\vdash_{\delta'} \mathrm{G}(\mathrm{v}_k^{a\rho})$; we will then have $\mathrm{G}(\mathrm{v}_k^{a\rho}) \mid \delta' \sqsubseteq h \mid \delta'$. For applying induction hypothesis to $(\phi, \mathrm{G}(\mathrm{v}_k^{a\rho}) \preccurlyeq h);\theta \vdash \mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho})) \preccurlyeq h$, we then require $\mathrm{G}(\mathrm{v}_k^{a\rho})\delta' \sqsubseteq h\delta'$, $\vdash_{\delta'} \mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho}))$ and $\mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho})) \mid \delta' \sqsubseteq h \mid \delta'$.

Lemma 18. *If $\vdash_{\delta'} \mathrm{STEP}(\overline{w})\ v_1$, $\vdash_{\delta'} v_2$ and $v_1 \mid \delta' = v_2 \mid \delta'$, then $\vdash_{\delta'} \mathrm{STEP}(\overline{w})\ v_2$ and $(\mathrm{STEP}(\overline{w})\ v_1) \mid \delta' = (\mathrm{STEP}(\overline{w})\ v_2) \mid \delta'$.*

Proof. By a straightforward induction on $\overline{w}$. ☐

By a straightforward induction on $\overline{w}$, we have $\vdash_{\delta'} \mathrm{BASE}(\overline{w}) = \mu x : a.e\rho$. Note also that inversion of the latter gives (1) $\vdash_{\delta'} \rho$, (2) $\rho\delta' \vdash a : \mathsf{V}$ and (3) $\forall d.\ (\rho\delta', x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta' = d) \vdash e : a\rho\delta'$.

Since $\vdash_{\delta'} \mathrm{STEP}(\overline{w})(\mathrm{BASE}(\overline{w})) = \mathrm{G}(\mu x : a.e\rho)$, $\vdash_{\delta'} \mathrm{v}_k^{a\rho}$ and $\mu x : a.e\rho \mid \delta' = \mathrm{v}_k^{a\rho} \mid \delta'$, Lemma 18 gives $\vdash_{\delta'} \mathrm{G}(\mathrm{v}_k^{a\rho})$ and $\overline{w} \mid \delta' = \mathrm{G}(\mathrm{v}_k^{a\rho}) \mid \delta'$. Thus induction hypothesis applies to $\phi;\theta \vdash \mathrm{G}(\mathrm{v}_k^{a\rho}){\uparrow} \preccurlyeq h{\uparrow}$, giving $\overline{w} \mid \delta' = \mathrm{G}(\mathrm{v}_k^{a\rho}) \mid \delta' \sqsubseteq h \mid \delta'$.

Lemma 19. *Forall $n \in \mathbb{N}$, we have $\vdash_{\delta'} \mathrm{F}^n(\mathrm{v}_k^{a\rho})$ and $(\mathrm{F}^n(\mathrm{v}_k^{a\rho})) \mid \delta' = \mathrm{v}_k^{a\rho} \mid \delta'$.*

Proof. Straightforward induction on n, using conditions (1–3). ☐

Now Lemmas 18 and 19 together give $\vdash_{\delta'} \mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho}))$ and $\mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho})) \mid \delta' \sqsubseteq h \mid \delta'$. The only condition missing from those required for applying induction hypothesis to $(\phi, \mathrm{G}(\mathrm{v}_k^{a\rho}) \preccurlyeq h);\theta \vdash \mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho})) \preccurlyeq h$ is now $\mathrm{G}(\mathrm{v}_k^{a\rho})\delta' \sqsubseteq h\delta'$, which we will keep as an assumption. Thus we obtain $\mathrm{G}(\mathrm{v}_k^{a\rho})\delta' \sqsubseteq h\delta' \implies \mathrm{G}(\mathrm{F}^n(\mathrm{v}_k^{a\rho}))\delta' \sqsubseteq h\delta'$.

Note that the latter is equivalent to $(\mathrm{G}\delta')\ (\mathrm{v}_k^{a\rho}\delta') \sqsubseteq h\delta' \implies (\mathrm{G}\delta')\ ((\mathrm{F}^n(\mathrm{v}_k^{a\rho}))\delta') \sqsubseteq h\delta'$, where $\mathrm{G}\delta'$ is defined by induction on $\overline{w}$ as follows.

$$\begin{aligned}(\mathrm{STEP}(\mu x : a.e\rho))\delta' &= \lambda d.\ d\\ (\mathrm{STEP}(\overline{w}\ v))\delta' &= \lambda d.\ ((\mathrm{STEP}(\overline{w}))\delta')\ (v\delta')\end{aligned}$$

Lemma 20. *Let $f = \lambda d.\ \mathsf{p}_{\mathsf{p}_{\mathsf{V}}\, a\rho\delta'}(e(\rho\delta', x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta' = d))$. Then $(\mathrm{F}^n(\mathrm{v}_k^{a\rho}))\delta' = f^n\ d$, where $f^0 = f$ and $f^{i+1} = f \circ f^i$.*

Proof. By induction on n.

– Case $n = 0$.

$$
\begin{array}{ll}
 & (\mathrm{F}^0(\mathrm{V}_k^{a\rho}))\delta' \\
= & \hfill \{\text{definition of } \mathrm{F}^0\} \\
 & e(\rho, x = \mathrm{V}_k^{a\rho})\delta' \\
= & \hfill \{\text{semantics of environments}\} \\
 & e(\rho\delta', x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta' = d) \\
= & \hfill \{\text{conditions (1–3)}\} \\
 & \mathsf{p}_{\mathsf{p}_{\mathsf{V}}\, a\rho\delta'}\, e(\rho\delta', x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta' = d) \\
= & \hfill \{\text{definition of } f^0\} \\
 & f^0 \ d
\end{array}
$$

– Case $n = m + 1$.

$$
\begin{array}{ll}
 & (\mathrm{F}^{m+1}(\mathrm{V}_k^{a\rho}))\delta' \\
= & \hfill \{\text{definition of } \mathrm{F}^{m+1}\} \\
 & e(\rho, x = \mathrm{F}^m(\mathrm{V}_k^{a\rho}))\delta' \\
= & \hfill \{\text{semantics of environments}\} \\
 & e(\rho\delta', x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta' = \mathrm{F}^m(\mathrm{V}_k^{a\rho})\delta') \\
= & \hfill \{\text{conditions (1–3), induction hypothesis}\} \\
 & \mathsf{p}_{\mathsf{p}_{\mathsf{V}}\, a\rho\delta'}\, e(\rho\delta', x : \mathsf{p}_{\mathsf{V}}\, a\rho\delta' = f^m \ d) \\
= & \hfill \{\text{definition of } f^{m+1}\} \\
 & f^{m+1} \ d
\end{array}
$$

$\square$

We now have $(\mathrm{G}\delta')\ (\mathsf{p}_{\mathsf{p}_{\mathsf{V}}\, a\rho\delta'}\, d) \sqsubseteq h\delta' \implies (\mathrm{G}\delta')\ (f^n \ d) \sqsubseteq h\delta'$, and thus $(\mathrm{G}\delta')\ d \sqsubseteq h\delta' \implies (\mathrm{G}\delta')\ (f^n \ d) \sqsubseteq h\delta'$. Note that the predicate $(\mathrm{G}\delta')\ d \sqsubseteq h\delta'$ is admissible (chain-complete) in d, and that $(\mathrm{G}\delta')\ \bot = \bot \sqsubseteq h\delta'$. Hence fixpoint induction applies, giving $(\mathrm{G}\delta')\ (\mathsf{fix}\,(f^n)) \sqsubseteq h\delta'$. Since $\mathsf{fix}\,(f^n) = \mathsf{fix} f = \mu x : a.e\rho\delta' = (\mathrm{BASE}(\overline{w}))\delta'$, we have $\overline{w}\delta' \sqsubseteq h\delta'$ and thus $\overline{w}\delta \sqsubseteq h\delta$, as required. (Note that $\overline{w} \mid \delta \sqsubseteq h \mid \delta$ was already shown above using Lemma 18.)

- Case $\phi; \theta \vdash w \preccurlyeq h$, where $\phi; \theta \vdash w\uparrow \preccurlyeq h\uparrow$, $w \xrightarrow{\mu}{}_{\theta}^{*} h_1$, $h \xrightarrow{\mu}{}_{\theta}^{*} h_2$ and $\phi; \theta \vdash h_1 \preccurlyeq h_2$. The desired result follows from Proposition 5 and induction hypotheses. (Note that the premise $\phi; \theta \vdash w\uparrow \preccurlyeq h\uparrow$ is only required if h is contained in the grammar w.)
- Case $\phi; \theta \vdash h \preccurlyeq w$, where $\phi; \theta \vdash h\uparrow \preccurlyeq w\uparrow$, $\phi; \theta \vdash w \preccurlyeq \perp\!\!\!\perp^{(w\uparrow)}$ and $\phi; \theta \vdash h \preccurlyeq \perp\!\!\!\perp^{(w\uparrow)}$. Immediate from induction hypotheses. (Note that the premise $\phi; \theta \vdash h\uparrow \preccurlyeq w\uparrow$ is only required if h is contained in the grammar w.)
- Case $\phi; \theta \vdash h_1 \preccurlyeq w$, where $\phi; \theta \vdash h_1\uparrow \preccurlyeq w\uparrow$, $w \xrightarrow{\mu}{}_{\theta}^{*} h_2$ and $\phi; \theta \vdash h_1 \preccurlyeq h_2$. Follows from Proposition 5 and induction hypotheses. (Note that the premise $\phi; \theta \vdash h_1\uparrow \preccurlyeq w\uparrow$ is only required if h_1 is contained in the grammar w.)

$\square$

B.9 Proof of Proposition 7

Let ρ be a syntactic environment, e an expression, t a value, and δ a valuation. Then

$$\vdash_\delta \theta \wedge \vdash_\delta \rho \;\wedge\; \theta;\rho \vdash e \implies \vdash_\delta e\rho$$

$$\vdash_\delta \theta \wedge \vdash_\delta \rho \wedge \vdash_\delta t : \mathsf{V} \;\wedge\; \theta;\rho \vdash e : t \implies \vdash_\delta e\rho : t\delta$$

$$\vdash_\delta \theta \wedge \vdash_\delta \rho \wedge \vdash_\delta \langle s\rangle\rho' \;\wedge\; \theta;\rho \models r : \langle s\rangle\rho' \implies \rho\delta \models r : \langle s\rangle\rho'\delta$$

$$\begin{pmatrix} \vdash_\delta \theta \wedge \vdash_\delta h : \mathsf{Data}\ \Delta \wedge \vdash_\delta a\rho : \mathsf{V} \wedge \\ (\forall c \in dom(b).\ c\Delta = \mathsf{Def}\ u) \;\wedge\; \theta;\rho \vdash h; b : a \end{pmatrix} \implies \Delta; \rho\delta \vdash h\delta; b : a\rho\delta$$

Proof. By induction on derivations.

- Case $\theta;\rho \vdash e : t$. By induction hypothesis on $\theta;\rho \vdash e$, we have $\vdash_\delta e\rho$. Now Proposition 6 applies to $();\theta \vdash e \mid \rho \preccurlyeq t$ and $();\theta \vdash t \preccurlyeq e \mid \rho$, giving $e \mid \rho\delta \sqsubseteq t\delta$ and $t\delta \sqsubseteq e \mid \rho\delta$. Thus $e \mid \rho\delta = t\delta$ as required for $\vdash_\delta e\rho : t\delta$.
- Cases $\theta;\rho \vdash x$ and $\theta;\rho \vdash \star$. Immediate.
- Case $\theta;\rho \vdash \mu x : a.e$. We need to show (1) $\rho\delta \vdash a : \mathsf{V}$ and (2) $\forall d.\ (\rho\delta, x : \mathsf{p_V}\,a\rho\delta = d) \vdash e : a\rho\delta$. By induction hypothesis on $\theta;\rho \vdash a : \star$, we have $\vdash_\delta a\rho : \mathsf{V}$, and thus (1) by inversion. For (2), let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. Freshness of $\mathrm{v}_k^{a\rho}$ implies (3) $\vdash_\delta \rho \iff \vdash_{\delta'} \rho$, (4) $\rho\delta = \rho\delta'$ and (5) $\vdash_\delta \theta \iff \vdash_{\delta'} \theta$. Assumption $\vdash_\delta \rho$ and conditions (1), (3) and (4) imply $\vdash_{\delta'} (\rho, x = \mathrm{v}_k^{a\rho})$. By the latter and conditions (1) and (5), induction hypothesis now applies to $\theta; (\rho, x = \mathrm{v}_k^{a\rho}) \vdash e : a\rho$, giving $\vdash_{\delta'} e(\rho, x = \mathrm{v}_k^{a\rho}) : a\rho\delta'$. By (4), inversion of the latter gives (2).
- Cases $\theta;\rho \vdash \lambda x : a.e$, $\theta;\rho \vdash \forall x : a_1.a_2$. Similar to the previous case.
- Case $\theta;\rho \vdash e_1\ e_2$. We prove the case where the second premise is $e_1 \mid \rho \xrightarrow{\mu}\!\!{}_\theta^{\!*}\ \lambda x : a.e \mid \rho'$. The other case is similar. We need to show (1) $\rho\delta \vdash e_1$, (2) $e_1 \mid \rho\delta = \mathsf{Prod}\ (e_2 \mid \rho\delta)\ f$ for some f, and (3) $\rho\delta \vdash e_2$. Induction hypothesis on $\theta;\rho \vdash e_1$ gives (1). Now Proposition 5 applies to the premise $e_1 \mid \rho \xrightarrow{\mu}\!\!{}_\theta^{\!*}\ \lambda x : a.e \mid \rho'$, giving (4) $\vdash_\delta \lambda x : a.e \mid \rho'$ and (5) $e_1 \mid \rho\delta = \mathsf{Prod}\ (\mathsf{p_V}\,a\rho'\delta)\ f$. Inversion of (4) gives $\vdash_\delta a\rho' : \mathsf{V}$. Now induction hypothesis applies to $\theta;\rho \vdash e_2 : a\rho'$, giving (3) and $e_2 \mid \rho\delta = a\rho'\delta = \mathsf{p_V}\,a\rho'\delta$. By combining the latter with (5), we obtain (2).
- Case $\theta;\rho \vdash e.x$. We need to show (1) $\rho\delta \vdash e$, (2) $e \mid \rho\delta = [xs : s]$ and (3) $x \in xs$. Induction hypothesis on $\theta;\rho \vdash e$ gives (1). Now Proposition 5 applies to the premise $e \mid \rho \xrightarrow{\mu}\!\!{}_\theta^{\!*}\ \langle s\rangle\rho'$, giving $e \mid \rho\delta = \langle s\rangle\rho'\delta$ and $x \in dom(s)$; thus (2) and (3) hold.
- Case $\theta;\rho \vdash \langle\bullet\rangle$. Immediate.

- Case $\theta; \rho \vdash \langle x = e, s\rangle$. We need to show (1) $\rho\delta \vdash e$ and (2) $(\rho\delta, x : e \mid \rho\delta = e\rho\delta) \vdash \langle s\rangle$. Induction hypothesis on $\theta; \rho \vdash e$ gives (1). Thus we also have $\vdash_\delta (\rho, x = e\rho)$, so that induction hypothesis applies to $\theta; (\rho, x = e\rho) \vdash \langle s\rangle$, giving $\vdash_\delta \langle s\rangle(\rho, x = e\rho)$. By inverting the latter, we obtain (2).

- Case $\theta; \rho \vdash \langle x : a, s\rangle$. We need to show (1) $\rho\delta \vdash a : \mathsf{V}$ and (2) $\forall d.\ (\rho\delta, x : \mathsf{p_V}\, a\rho\delta = d) \vdash \langle s\rangle$. Induction hypothesis on $\theta; \rho \vdash a : \star$ gives $\vdash_\delta a\rho : \mathsf{V}$, and by inversion we obtain (1). For (2), let $d \in \mathsf{D}$ and consider $\delta' = (\delta, k = d)$. As shown in previous cases, freshness of $\mathrm{v}_k^{a\rho}$ and condition (1) imply that $\vdash_{\delta'} (\rho, x = \mathrm{v}_k^{a\rho})$. Now induction hypothesis applies to $\theta; (\rho, x = \mathrm{v}_k^{a\rho}) \vdash \langle s\rangle$, giving $\vdash_{\delta'} \langle s\rangle(\rho, x = \mathrm{v}_k^{a\rho})$. Inverting the latter and noting that $\rho\delta = \rho\delta'$, we obtain (2).

- Case $\theta; \rho \vdash \langle r :: a\rangle$. We need to show (1) $\rho\delta \vdash a : \mathsf{V}$, (2) $a\rho\delta = [xs : s]$ and (3) $\rho\delta \models r : [xs : s]$. Induction hypothesis on $\theta; \rho \vdash a : \star$ gives $\vdash_\delta a\rho : \mathsf{V}$, and thus (1) by inversion. Now Proposition 5 applies to the premise $a\rho \xrightarrow{\mu}{}_{\theta}^{*} \langle s\rangle\rho'$, giving (4) $\vdash_\delta \langle s\rangle\rho'$, and $a\rho\delta = \langle s\rangle\rho'\delta = [xs : s]$ as required for (2). By (4), induction hypothesis now applies to $\theta; \rho \models r : \langle s\rangle\rho'$, and we obtain (3).

- Case $\theta; \rho \models \bullet : \langle\bullet\rangle\rho'$. Immediate.

- Case $\theta; \rho \models r : \langle x = e, s\rangle\rho'$. We need to show that $\rho\delta \models r : \langle x = e, s\rangle\rho'\delta$. By inversion, and since $\langle x = e, s\rangle\rho'\delta = [x : (e \mid \rho'\delta) = e\rho'\delta, \langle s\rangle(\rho'\delta, x : (e \mid \rho'\delta) = e\rho'\delta)]$, it suffices to show $\rho\delta \models r : \langle s\rangle(\rho'\delta, x : (e \mid \rho'\delta) = e\rho'\delta)$. This follows immediately from the induction hypothesis on the premise $\theta; \rho \models r : \langle s\rangle(\rho', x = e\rho')$.

- Case $\theta; \rho \models (x = e, r) : \langle x : a, s\rangle\rho'$. We need to show $\rho\delta \models (x = e, r) : \langle x : a, s\rangle\rho'\delta$. By inversion, and since $\langle x : a, s\rangle\rho'\delta = [x : \mathsf{p_V}\, a\rho'\delta, \lambda d.\langle s\rangle(\rho'\delta, x : \mathsf{p_V}\, a\rho'\delta = d)]$, it suffices to show (1) $\rho\delta \vdash e : \mathsf{p_V}\, a\rho'\delta$ and (2) $\rho\delta \models r : \langle s\rangle(\rho'\delta, x : \mathsf{p_V}\, a\rho'\delta = e\rho\delta)$. By inversion of the assumption $\vdash_\delta \langle x : a, s\rangle\rho'$, we obtain (3) $\vdash_\delta a\rho' : \mathsf{V}$ and (4) $\forall d.\ (\rho'\delta, x : \mathsf{p_V}\, a\rho'\delta = d) \vdash \langle s\rangle$. By (3), induction hypothesis now applies to $\theta; \rho \vdash e : a\rho'$, giving (5) $\vdash_\delta e\rho : a\rho'\delta = \mathsf{p_V}\, a\rho'\delta$, and by inversion we obtain (1). By (1) and (4), we have $\vdash_\delta (\rho', x = e\rho)$, so that the induction hypothesis applies to the premise $\theta; \rho \models r : \langle s\rangle(\rho', x = e\rho)$, giving $\rho\delta \models r : \langle s\rangle(\rho', x = e\rho)\delta$, which together with (1) implies (2).

- Case $\theta; \rho \vdash c\ \ e\ \ a$. We need to show (1) $\rho\delta \vdash a : \mathsf{V}$, (2) $a\rho\delta = \mathsf{Data}\ \Delta$, (3) $c\Delta = \mathsf{Def}\ u$ for some u, and (4) $\rho\delta \vdash e : u$. Induction hypothesis on $\theta; \rho \vdash a : \star$ gives $\vdash_\delta a\rho : \mathsf{V}$, and thus (1) by inversion. Now Proposition 5 applies to the premise $a\rho \xrightarrow{\mu}{}_{\theta}^{*} \{d\}\rho'$, giving (5) $\vdash_\delta \{d\}\rho'$ and (6) $a\rho\delta = \{d\}\rho'\delta = \mathsf{Data}\ \Delta$. By the premise $c \in dom(d)$, we now have (3). By (3), (5) and (6), we have $\vdash_\delta \pi_c(\{d\}\rho') : \mathsf{V}$, so that induction hypothesis applies to $\theta; \rho \vdash e : \pi_c(\{d\}\rho')$, giving $\vdash_\delta e\rho : \pi_c(\{d\}\rho')\delta$. Since $\pi_c(\{d\}\rho')\delta = \pi_c(\mathsf{Data}\ \Delta) = u$ by (3), we thus have (4).

- Case $\theta; \rho \vdash \{\bullet\}$. Immediate.

- Case $\theta;\rho \vdash \{c\ a, d\}$. We need to show (1) $\rho\delta \vdash a : \mathsf{V}$ and (2) $\rho\delta \vdash \{d\}$. Induction hypothesis on $\theta;\rho \vdash a : \star$ gives $\vdash_\delta a\rho : \mathsf{V}$, and thus (1) by inversion. By induction hypothesis on $\theta;\rho \vdash \{d\}$ we have (2).

- Case $\theta;\rho \vdash \mathit{case}\ a\ b\ e$. We need to show (1) $\rho\delta \vdash a : \mathsf{V}$, (2) $\rho\delta \vdash e$, (3) $e \mid \rho\delta = \mathsf{Data}\ \Delta$ and (4) $\Delta;\rho\delta \vdash e\rho\delta; b : a\rho\delta$. Induction hypothesis on $\theta;\rho \vdash a : \star$ gives $\vdash_\delta a\rho : \mathsf{V}$, and thus (1) by inversion. Similarly, induction hypothesis on $\theta;\rho \vdash e$ gives (2). Now Proposition 5 applies to the premise $e \mid \rho \xrightarrow[\theta]{\mu}\!\!{}^* \{d\}\rho'$, giving (5) $\vdash_\delta \{d\}\rho'$ and $e \mid \rho\delta = \{d\}\rho'\delta = \mathsf{Data}\ \Delta$, as required for (3). Proposition 5 also applies to the premise $e\rho \xrightarrow[\theta]{\mu}\!\!{}^* h$, giving $\vdash_\delta h$, $e\rho\delta = h\delta$ and $e \mid \rho\delta = h \mid \delta = \mathsf{Data}\ \Delta$. Note that we have $\forall c \in \mathit{dom}(b).\ c\Delta = \mathsf{Def}\ u$ by the premise $\forall c \in \mathit{dom}(b).\ c \in \mathit{dom}(d)$. For (4), it thus suffices to show $\Delta;\rho\delta \vdash h\delta; b : a\rho\delta$, which now follows from induction hypothesis on $\theta;\rho \vdash h; b : a$.

- Case $\theta;\rho \vdash \perp\!\!\!\perp^t; b : a$. By a straightforward induction on b, using the assumption $\forall c \in \mathit{dom}(b).\ c\Delta = \mathsf{Def}\ u$.

- Case $\theta;\rho \vdash (c\ e\ a)\rho'; \bullet : a$. Immediate.

- Case $\theta;\rho \vdash (c'\ e'\ a')\rho'; (c\ x.e, b) : a$ where $c \neq c'$. We need to show (1) $\forall d.\ ((c'\ e'\ a')\rho'\delta = \mathsf{Inj}_c\ d \implies (\rho\delta, x : u = d) \vdash e : a\rho\delta)$, (2) $c\Delta = \mathsf{Def}\ u$ and (3) $\Delta;\rho\delta \vdash (c'\ e'\ a')\rho'\delta; b : a\rho\delta$. Condition (1) holds vacuously, since $(c'\ e'\ a')\rho'\delta = \mathsf{Inj}_c\ d$ does not hold when $c \neq c'$. Condition (2) holds by assumption. Condition (3) holds by induction hypothesis on $\theta;\rho \vdash (c'\ e'\ a')\rho'; b : a$.

- Case $\theta;\rho \vdash (c\ e'\ a')\rho'; (c\ x.e, b) : a$. We need to show (1) $\forall d.\ ((c\ e'\ a')\rho'\delta = \mathsf{Inj}_c\ d \implies (\rho\delta, x : u = d) \vdash e : a\rho\delta)$, (2) $c\Delta = \mathsf{Def}\ u$ and (3) $\Delta;\rho\delta \vdash (c\ e'\ a')\rho'\delta; b : a\rho\delta$. Note that inversion of the assumption $\vdash_\delta (c\ e'\ a')\rho'$ gives (4) $\rho'\delta \vdash a' : \mathsf{V}$, (5) $a'\rho'\delta = \mathsf{Data}\ \Delta$, (6) $c\Delta = \mathsf{Def}\ u$ and (7) $\rho'\delta \vdash e' : u$, as well as (8) $\vdash_\delta e'\rho'$. We then have

$$
\begin{aligned}
& (c\ e'\ a')\rho'\delta \\
= \quad & && \{\text{semantics of expressions}\} \\
& \mathsf{p}_{\mathsf{pV}\, a'\rho'\delta}(\mathsf{Inj}_c\ (e'\rho'\delta)) \\
= \quad & && \{(4)\} \\
& \mathsf{p}_{a'\rho'\delta}(\mathsf{Inj}_c\ (e'\rho'\delta)) \\
= \quad & && \{(5)\} \\
& \mathsf{p}_{\mathsf{Data}\ \Delta}(\mathsf{Inj}_c\ (e'\rho'\delta)) \\
= \quad & && \{(6)\} \\
& \mathsf{Inj}_c\ \mathsf{p}_u(e'\rho'\delta) \\
= \quad & && \{(7)\} \\
& \mathsf{Inj}_c\ (e'\rho'\delta)
\end{aligned}
$$

By assumption $\vdash_\delta a\rho : \mathsf{V}$, induction hypothesis now applies to $\theta; (\rho, x = e'\rho') \vdash e : a\rho$, giving $\vdash_\delta e(\rho, x = e'\rho') : a\rho\delta$. Inversion of the latter gives $(\rho\delta, x : e' \mid \rho'\delta = e'\rho'\delta) \vdash e : a\rho\delta$, which combined with (7) gives (1). Condition (2) holds by assumption. Condition (3) follows by straightforward induction on b, using the assumption $\forall c \in dom(b).\ c\Delta = \mathsf{Def}\ u$, and using the fact that $c \notin dom(b)$ because we do not allow duplicate constructors in branches.

- Case $\theta; \rho \vdash \widehat{g}; b : a$. We need to show that $\Delta; \rho\delta \vdash \widehat{g}\delta; b : a\rho\delta$. By induction on $\widehat{g}$, we have that (1) $\vdash_\delta w = \text{SCRUTINEE}(\widehat{g})$. Thus Proposition 5 applies to $w{\uparrow} \xrightarrow[\theta]{\mu}\!\triangleright^* \{d\}\rho'$, giving (2) $\vdash_\delta \{d\}\rho'$ and (3) $w \mid \delta = \{d\}\rho'\delta = \mathsf{Data}\ \Delta'$. Since $w\delta : \mathsf{Data}\ \Delta'$, we must have either $w\delta = \bot$ or $w\delta = \mathsf{Inj}_c\ d'$, for some $c \in dom(d)$ and $d' \in \mathsf{D}$.
  - Case $w\delta = \bot$. The desired conclusion follows as in the case for $\theta; \rho \vdash \perp\!\!\!\perp^t; b : a$.
  - Case $w\delta = \mathsf{Inj}_c\ d'$, for some $c \in dom(d)$ and $d' \in \mathsf{D}$. Consider $\delta' = (\delta, k = d')$. Freshness of $\mathrm{v}_k^{\pi_c(\{d\}\rho')}$ implies that $\vdash_\delta \theta \iff \vdash_{\delta'} \theta$, $\vdash_\delta \rho' \iff \vdash_{\delta'} \rho'$ and $\rho'\delta = \rho'\delta'$. To show that $\vdash_{\delta'} \theta_c$, we need to show (4) $\vdash_{\delta'} (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho')$, (5) $w\delta' = (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho')\delta'$ and (6) $w \mid \delta' = (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho') \mid \delta'$. Condition (4) follows from (2), (3), assumption $c \in dom(d)$ and freshness of $\mathrm{v}_k^{\pi_c(\{d\}\rho')}$. For (5) and (6) we have

$$
\begin{array}{cl}
 & (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho')\delta' \\
= & \hfill \{\text{semantics of expressions}\} \\
 & \mathsf{p}_{\mathsf{pV}(y(x=\mathrm{v}_k^{\pi_c(\{d\}\rho')}, y=\{d\}\rho')\delta')}(\mathsf{Inj}_c\ (x(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho')\delta')) \\
= & \hfill \{(3) \text{ and freshness of } \mathrm{v}_k^{\pi_c(\{d\}\rho')}\} \\
 & \mathsf{p}_{(\mathsf{Data}\ \Delta')}(\mathsf{Inj}_c\ (\mathsf{p}_{\mathsf{pV}(\pi_c(\{d\}\rho')\delta')}\, d')) \\
= & \hfill \{c\Delta' = \mathsf{Def}\ u\} \\
 & \mathsf{p}_{(\mathsf{Data}\ \Delta')}(\mathsf{Inj}_c\ (\mathsf{p}_u\, d')) \\
= & \hfill \{c\Delta' = \mathsf{Def}\ u\} \\
 & \mathsf{p}_{(\mathsf{Data}\ \Delta')}(\mathsf{p}_{(\mathsf{Data}\ \Delta')}(\mathsf{Inj}_c\ d')) \\
= & \hfill \{\mathsf{Inj}_c\ d' = w\delta : \mathsf{Data}\ \Delta'\} \\
 & w\delta
\end{array}
$$

and

$$
\begin{array}{ll}
& (c\ x\ y)(x = \mathrm{v}_k^{\pi_c(\{d\}\rho')}, y = \{d\}\rho') \mid \delta' \\
= & \hfill \{\text{semantics of expressions}\} \\
& \mathsf{pv}(\{d\}\rho'\delta') \\
= & \hfill \{\text{freshness of } \mathrm{v}_k^{\pi_c(\{d\}\rho')}\} \\
& \mathsf{pv}(\{d\}\rho'\delta) \\
= & \hfill \{(3)\} \\
& w \mid \delta
\end{array}
$$

respectively. We have thus shown that $\vdash_{\delta'} \theta_c$, for $c \in dom(d)$. Now Proposition 5 applies to $\widehat{g} \xrightarrow[\theta]{\mu}\!\!\triangleright^* h$, giving $\vdash_{\delta'} h$, $h\delta' = \widehat{g}\delta'$ and $h \mid \delta' = \widehat{g} \mid \delta'$, by freshness of $\mathrm{v}_k^{\pi_c(\{d\}\rho')}$. Hence induction hypothesis now applies to $\theta_c; \rho \vdash h; b : a$, giving $\Delta; \rho\delta' \vdash h\delta'; b : a\rho\delta'$, which is equivalent to $\Delta; \rho\delta \vdash \widehat{g}\delta; b : a\rho\delta$ by freshness of $\mathrm{v}_k^{\pi_c(\{d\}\rho')}$, as required.

$\square$

B.10 Proof of Theorem 8

Let a and e be expressions. Then for all $\nu \in \mathsf{Env}$

$$();() \vdash a : \star \quad \wedge \quad ();() \vdash e : a() \quad \Longrightarrow \quad e\nu : a\nu$$

Proof. The theorem follows from Proposition 7 and Lemma 2, as well as the following observations: (1) if $\theta; \rho \vdash e_0$ then free variables of e_0 are in $dom(\rho)$, hence e and a above are closed; (2) $e_0\nu = e_0\nu'$ if ν and ν' agree on free variables of e_0, so in particular $e\nu$ and $a\nu$ are constant in ν and equal to $e\bot_{\mathsf{Env}}$ and $a\bot_{\mathsf{Env}}$, respectively. $\square$