Specification of Core Agda*
Subtitle†

FIRST1 LAST1‡, Institution1
FIRST2 LAST2§, Institution2a and Institution2b

This document specifies the abstract syntax, evaluation rules, and typing rules of Core Agda, the basic type theory underlying Agda.

CCS Concepts: •**Software and its engineering** → **General programming languages;** •**Social and professional topics** → *History of programming languages;*

Additional Key Words and Phrases: Agda, dependent types, specification

ACM Reference format:
First1 Last1 and First2 Last2. 2017. Specification of Core Agda. *PACM Progr. Lang.* 1, 1, Article 1 (January 2017), **??** pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Agda 2 has been developed since 2005, and been released 2007. So far, no specification has been given. This document attempts to specify the core components of Agda.

2 SYNTAX

2.1 Terms

We distinguish global names f, F, D, c, R, π bound in the signature Σ from local variables x, y, z bound in typing contexts Γ. Local variables are represented by de Bruijn indices in the implementation, but for the sake of readability we stick to a named presentation here. We silently rename bound variables to avoid clashes with free variables (Barendregt convention). We write $x\#E$ ("x fresh for E") to express that x is a fresh variable with regard to syntactic entity E.

For orientation of the reader, we use different letters to represent different purposes of global names. However, they share the same name space and are not distinguished syntactically.

D	data type name
R	record type name
F	data type or record type name
f	function name
π	projection name (overloaded)
c	data/record constructor name (overloaded)

*with title note
†with subtitle note
‡with author1 note
§with author2 note

with paper note.
2017. 2475-1421/2017/1-ART1 $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

Projection and constructor names can be overloaded and are resolved by the type checker. We write $R.\pi$ for a resolved record projection name, $R.c$ for a resolved record constructor name and $D.c$ for a resolved data constructor name.

Terms use a spine form for eliminations and thus are kept β and projection normal form. This means that terms cannot be a β-redex $(\lambda x.\, v)\, u$ nor a projection of a record value $c_{\vec{\pi}}\, \vec{v}\, .\pi$.

$$
\begin{array}{rcll}
t, u, v & ::= & x\ \bar{e} & \text{variables (eliminated by } \bar{e}) \\
& \mid & f\ \bar{e} & \text{defined function (eliminated by } \bar{e}\text{), includes postulates} \\
& \mid & c\ \bar{v} & \text{data constructor applied to arguments } \bar{v} \\
& \mid & c_{\vec{\pi}}\ \bar{v} & \text{record constructor with fields } \vec{\pi} \text{ applied to arguments } \bar{v} \\
& \mid & \lambda x.\, v & \text{lambda abstraction} \\
& \mid & F\ \bar{v} & \text{data/record type name applied to arguments } \bar{v} \\
& \mid & (x : U) \to V & \text{dependent function type} \\
& \mid & s & \text{Sort}
\end{array}
$$

We use uppercase letters T, U, V for terms that stand for types.

Eliminations e exists for functions and records. Functions are eliminated by application, records by projection.

$$
\begin{array}{rcll}
e & ::= & @u & \text{application to term } u \\
& \mid & .\pi & \text{taking projection } \pi
\end{array}
$$

Other forms of elimination, like definiting a function by cases over some data structure, need pattern matching, which is supported in function declarations, but not in terms directly.

Sorts s are the types of types. Here, we model a predicative hierarchy of universes $\mathsf{Set}_0 : \mathsf{Set}_1 : \mathsf{Set}_2 : \ldots$.

$$
\begin{array}{rcll}
s & ::= & \mathsf{Set}_n & \text{universe of types of level } n \in \mathbb{N}
\end{array}
$$

We define *sort supremum* $\boxed{s \sqcup s'}$ by $\mathsf{Set}_i \sqcup \mathsf{Set}_j = \mathsf{Set}_{\max(i,j)}$.

2.2 Telescopes and patterns

Telescopes Δ are like typing contexts Γ (see below), but right-associative. Telescopes are used e. g. for parameter lists in declarations.

$$
\begin{array}{rcll}
\Delta & ::= & \cdot & \text{empty telescope} \\
& \mid & (x : T)\Delta & \text{non-empty telescope; } \Delta \text{ can depend on } x
\end{array}
$$

Notation: iterated function types $\boxed{\Delta \to T}$, defined by recursion on Δ:

$$
\begin{array}{rcl}
\cdot \to T & = & T \\
(x : U)\Delta \to T & = & (x : U) \to (\Delta \to T)
\end{array}
$$

Patterns p are used for definition by cases. Patterns are made from variables x and data constructors c, but can also contain arbitrary terms u that do not bind any variables. The latter form is called *inaccessible* pattern, or, due to the concrete syntax used in Agda, *dot* pattern.

$$
\begin{array}{rcll}
p & ::= & x & \text{variable pattern} \\
& \mid & c\ \bar{p} & \text{constructor pattern} \\
& \mid & \lfloor u \rfloor & \text{inaccessible pattern (aka dot pattern)}
\end{array}
$$

Copatterns are to eliminations what patterns are to arguments. We match eliminations against copatterns like we match arguments against patterns.

q	$::=$	$@p$	application pattern
	$\mid$	$.\pi$	projection pattern

2.3 Declarations

Agda has three main *declaration* forms that introduce global names into the signature Σ. Function declarations introduce global functions f defined by pattern matching. A function declaration consists of a type signature ts and a list of function clauses $\vec{cl}$. The signature has to preceed the clauses, but between them other declarations are allowed, in order to facilitate mutually recursive definitions. Data (type) declarations consist of a data signature ds and a data definition dd, which have to appear in this order but can also be appart, to realize inductive-recursive definitions, for instance. Similarly record (type) declarations consist of a record signature rs and a record declaration rd.

d	$::=$	ts	type signature
	$\mid$	cl	function clause
	$\mid$	ds	data signature
	$\mid$	dd	data definition
	$\mid$	rs	record signature
	$\mid$	rd	record definition.

Type signatures consist of a global name f for the function and its type T.

$$ts \;::=\; f : T$$

A *function clause* for f consists of a copattern spine $\vec{q}$ and a right hand side rhs. The pattern variables of $\vec{q}$ are bound in context Γ. The right hand side is either empty or has evidence that the case is impossible.

cl	$::=$	$\Gamma \rhd f\ \bar{q} : U\ = rhs$	
rhs	$::=$	t	term (clause body)
	$\mid$	$\mathrm{absurd}\,\vec{x}$	$(\vec{x} \neq \emptyset)$ absurd clause (each variable x_i has empty type)

A *data signature* consists of a name D for the data type, a list of its parameters Δ, a list of itsindices Δ', and its target sort s.

$$\textbf{data}\ D\ \Delta : \Delta' \to s$$

Data definitions repeat the parameter telescope Δ, since parameters will be mentioned in the types T of the constructors c.

$$\textbf{data}\ D\ \Delta\ \textbf{where}\ \overline{c : T}$$

*Record signature*s carry, in contrast to data signatures, no index telescope Δ', as Agda does not support indexed record types.

$$\textbf{record}\ R\ \Delta : s$$

Record declarations supply a record constructor name c and a list of fields π with their types T.

$$\begin{array}{l}\textbf{record}\ R\ \Delta\ \textbf{where}\\ \quad \textbf{constructor}\ c\\ \quad \textbf{field}\ \overline{\pi : T}\end{array}$$

Declaration checking moves declarations to the signature Σ, but not literally, but in a refined form. A signature is a list of signature declarations (used internally).

$$\begin{array}{lcl} \Sigma & ::= & \overline{d_\Sigma} \\ d_\Sigma & ::= & \text{data } D\ \Delta : \Delta' \to s \text{ where } \overline{c : T} \\ & | & \text{record } R\ \Delta : s \text{ where } c : T\text{; projections } \overline{\pi : T} \\ & | & \text{function } f : T \text{ where } \overline{cl} \end{array}$$

TODO: UPDATE SIGNATURE DECLARATIONS.

3 TYPING RULES

3.1 Auxiliary judgements

Hereditary substitution $\boxed{u[\sigma] = v}$ is defined simultaneously with active elimination (see below).

$$\frac{\sigma(x) \mathbin{!} \overline{e}[\sigma] = v}{(x\overline{e})[\sigma] = v} \qquad \frac{\overline{e}[\sigma] = \overline{e}'}{(f\overline{e})[\sigma] = f\overline{e}'} \qquad \frac{\overline{v}[\sigma] = \overline{v}'}{(h\overline{v})[\sigma] = h\overline{v}'}\ h ::= F \mid c \mid c_{\overline{\pi}} \qquad \frac{v[\sigma] = v'}{(\lambda x.\, v)[\sigma] = \lambda x.\, v'}\ x\#\sigma$$

$$\frac{U[\sigma] = U' \qquad V[\sigma] = V'}{((x : U) \to V)[\sigma] = (x : U') \to V'}\ x\#\sigma \qquad \frac{}{s[\sigma] = s}$$

Active elimination $\boxed{t \mathbin{!} \overline{e} = v}$.

$$\frac{v[u/x] \mathbin{!} \overline{e} = v'}{\lambda x.v \mathbin{!} @u\overline{e} = v'} \qquad \frac{v_i \mathbin{!} \overline{e} = v'}{c_{\overline{\pi}}\overline{v} \mathbin{!} .\pi_i\overline{e} = v'} \qquad \frac{}{t \mathbin{!} \cdot = t} \qquad \frac{}{x\overline{e} \mathbin{!} \overline{e}' = x\overline{e}\overline{e}'} \qquad \frac{}{f\overline{e} \mathbin{!} \overline{e}' = f\overline{e}\overline{e}'}$$

Note that since constructor and data and record type terms are fully applied, we do not need to give rules for adding further applications to these. Of course, function type and sort expressions cannot be applied either.

Function type application $\boxed{T \mathbin{!!} \vec{u} = U}$.

$$\frac{}{T \mathbin{!!} . = T} \qquad \frac{V[u/x] \mathbin{!!} \vec{u} = T}{(x : U) \to V \mathbin{!!} u\vec{u} = T}$$

Signature lookup $\boxed{\Sigma \vdash z : T}$.

$$\frac{\text{function } f : T \text{ where } \overline{cl} \in \Sigma}{\Sigma \vdash f : T}$$

$$\frac{\text{data } D\ \Delta : \Delta' \to s \text{ where } \overline{c : T} \in \Sigma}{\Sigma \vdash D : \Delta \to \Delta' \to s} \qquad \frac{\text{data } D\ \Delta : \Delta' \to s \text{ where } \overline{c : T} \in \Sigma}{\Sigma \vdash D.c_i : \Delta \to T_i}$$

$$\frac{\text{record } R\ \Delta : s \text{ where } c : T\text{; projections } \overline{\pi : T} \in \Sigma}{\Sigma \vdash R : \Delta \to s}$$

$$\frac{\text{record } R\ \Delta : s \text{ where } c : T\text{; projections } \overline{\pi : T} \in \Sigma}{\Sigma \vdash R.c : \Delta \to T} \qquad \frac{\text{record } R\ \Delta : s \text{ where } c : T\text{; projections } \overline{\pi : T} \in \Sigma}{\Sigma \vdash R.\pi_i : \Delta \to T_i}$$

3.2 Typing judgements for expressions

Typing contexts.

$$
\begin{array}{lcl}
\Gamma & ::= & \cdot \mid \Gamma, x : T \\
\Delta & ::= & \cdot \mid x : T, \Delta
\end{array}
$$

Well-typed contexts $\boxed{\vdash_\Sigma \Gamma}$.

$$
\frac{}{\vdash_\Sigma \cdot} \qquad \frac{\vdash_\Sigma \Gamma \qquad \Gamma \vdash_\Sigma T : s}{\vdash_\Sigma \Gamma, x : T}\; x \notin \mathrm{dom}(\Gamma)
$$

Well-typed telescopes $\boxed{\Gamma \vdash_\Sigma \Delta}$.

$$
\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \cdot} \qquad \frac{\Gamma \vdash_\Sigma T : s \qquad \Gamma, x : T \vdash_\Sigma \Delta}{\Gamma \vdash_\Sigma (x : T)\Delta}
$$

Spine typing $\boxed{\Gamma \mid u : U \vdash_\Sigma \bar{e} : T}$ (u is neutral).

$$
\frac{}{\Gamma \mid u : U \vdash_\Sigma \cdot : U}
$$

$$
\frac{\Gamma \vdash_\Sigma u : U \qquad \Gamma \mid t @ u : T[u/x] \vdash_\Sigma \bar{e} : V}{\Gamma \mid t : \Pi x : U.T \vdash_\Sigma @u\bar{e} : V} \qquad \frac{\Sigma \vdash R.\pi = T \qquad T \mathbin{!!} (\vec{u}, t) = U \qquad \Gamma \mid t.\pi : U \vdash_\Sigma \bar{e} : V}{\Gamma \mid t : R\overline{u} \vdash_\Sigma .\pi\bar{e} : V}
$$

$$
\frac{\Gamma \mid u : U \vdash_\Sigma \vec{e} : V \qquad \Gamma \vdash_\Sigma U = U' : s}{\Gamma \mid u : U' \vdash_\Sigma \vec{e} : V}
$$

Term typing $\boxed{\Gamma \vdash_\Sigma t : T}$

$$
\frac{(x : U) \in \Gamma \qquad \Gamma \mid x : U \vdash_\Sigma \bar{e} : T}{\Gamma \vdash_\Sigma x\bar{e} : T} \qquad \frac{\Sigma \vdash f : T \qquad \Gamma \mid f : T \vdash_\Sigma \vec{e} : V}{\Gamma \vdash_\Sigma f\vec{e} : V}
$$

$$
\frac{\Sigma \vdash D.c : T \qquad T \mathbin{!!} \vec{u} = T' \qquad \Gamma \mid c : T' \vdash_\Sigma \vec{v} : D\,\vec{u}\,\vec{t}}{\Gamma \vdash_\Sigma c\,\vec{v} : D\,\vec{u}\,\vec{t}} \qquad \frac{\Sigma \vdash z : T \qquad \Gamma \mid z : T \vdash_\Sigma \vec{u} : V}{\Gamma \vdash_\Sigma z\,\vec{u} : V}\; z ::= D \mid R
$$

$$
\frac{\Gamma, x{:}U \vdash_\Sigma v : V}{\Gamma \vdash_\Sigma \lambda x.v : (x : U) \to V} \qquad \frac{\Gamma \vdash_\Sigma U : s \qquad \Gamma, x{:}U \vdash_\Sigma V : s'}{\Gamma \vdash_\Sigma (x : U) \to V : s \sqcup s'}
$$

Conversion $\boxed{\Gamma \vdash_\Sigma t = t' : T}$.

$$
\frac{f\,\overline{q} = t \in \Sigma \qquad (\overline{\lceil q \rceil})[\sigma] = \overline{e} \qquad t[\sigma] = v \qquad \Gamma \vdash_\Sigma f\,\overline{e} : T \qquad \Gamma \vdash_\Sigma v : T}{\Gamma \vdash_\Sigma f\overline{e} = v : T}
$$

$$
\frac{\Gamma \vdash_\Sigma t : (x : U) \to V}{\Gamma \vdash_\Sigma t = \lambda x.t\,x : (x : U) \to V} \qquad \frac{\Gamma \vdash_\Sigma t : R\,\vec{u} \qquad \Sigma \vdash R.c_{\vec{\pi}} : T}{\Gamma \vdash_\Sigma t = c_{\vec{\pi}}\,\overline{t.\pi} : R\,\vec{u}}
$$

And many boring rules (equivalence, congruence) and rules for elimination equality $\boxed{\Gamma \mid t : T \vdash_\Sigma \vec{e} = \vec{e}' : V}$

3.3 Type emptiness

$\boxed{\Gamma \vdash t \# t' : T}$ In context Γ, terms t and t' of type T cannot ever be unified.

$$\frac{\Gamma \vdash c\vec{v} : T \qquad \Gamma \vdash c'\vec{v}' : T}{\Gamma \vdash c\vec{v} \# c'\vec{v}' : T}\; c \neq c' \qquad \frac{\Gamma \vdash t \# t' : T}{\Gamma \vdash t' \# t : T} \qquad \frac{\Gamma \vdash t \# t' : T \qquad \Gamma \vdash t' = t'' : T}{\Gamma \vdash t \# t'' : T}$$

$$\frac{\Sigma \vdash c : \Delta \to \Delta' \to D\,\Delta\,\vec{u}' \qquad \Delta'[\vec{t}/\Delta] = \Delta'' \qquad \Gamma \vdash \vec{v} \# \vec{v}' : \Delta''}{\Gamma \vdash c\vec{v} \# c\vec{v}' : D\,\vec{t}\,\vec{u}} \qquad \frac{\Gamma \vdash t \# t' : T \qquad \Gamma \vdash T = T' : s}{\Gamma \vdash t \# t' : T'}$$

$\boxed{\Gamma \vdash \vec{v} \# \vec{v}' : \Delta}$ In context Γ, argument lists $\vec{v}$ and $\vec{v}'$ of telescope Δ cannot ever be unified.

$$\frac{\Gamma \vdash v \# v' : T}{\Gamma \vdash v\,\vec{v} \# v'\,\vec{v}' : (x\!:\!T)\Delta}$$

$$\frac{\Gamma \vdash v = v' : T \qquad \Delta[v/x] = \Delta' \qquad \Gamma \vdash \vec{v} \# \vec{v}' : \Delta'}{\Gamma \vdash v\vec{v} \# v'\vec{v}' : (x\!:\!T)\Delta} \qquad \frac{x \notin \mathsf{FV}(\Delta) \qquad \Gamma \vdash \vec{v} \# \vec{v}' : \Delta}{\Gamma \vdash v\vec{v} \# v'\vec{v}' : (x\!:\!T)\Delta}$$

$\boxed{\Gamma \vdash T \not\lesssim U}$

$$\frac{\Delta'[\vec{u}/\Delta] = \Delta'' \qquad \vec{v}'[\vec{u}/\Delta] = \vec{v}'' \qquad \Gamma, \Delta'' \vdash \vec{v}'' \# \vec{v} : \Delta''}{\Gamma \vdash \Delta \to \Delta' \to D\,\Delta\,\vec{v}' \not\lesssim D\,\vec{u}\,\vec{v}}$$

$\boxed{\Gamma \vdash T \text{ empty}}$: In context Γ, data type T has no canonical inhabitants.

$$\frac{\forall c.\ \Sigma \vdash D.c : U \Rightarrow \Gamma \vdash U \not\lesssim T}{\Gamma \vdash T \text{ empty}} \qquad \frac{\Gamma \vdash T \text{ empty} \qquad \Gamma \vdash T = T' : s}{\Gamma \vdash T' \text{ empty}}$$

3.4 Declaration typing

(Co)pattern variables $\boxed{PV(p) = \bar{x}}$

$$PV(x) = \{x\} \qquad PV(\lceil v \rceil) = \emptyset \qquad PV(c\,\bar{p}) = PV(\bar{p})$$
$$PV(\bar{p}) = \biguplus_i PV(p_i)$$
$$PV(.\pi) = \emptyset \qquad PV(@p) = PV(p)$$
$$PV(\bar{q}) = \biguplus_i PV(q_i)$$

Embedding of (co)patterns into terms $\boxed{\lceil p \rceil = t}$

$$\lceil x \rceil = x \qquad \lceil \lfloor v \rfloor \rceil = v \qquad \lceil c\,\bar{p} \rceil = c\,\lceil \bar{p} \rceil$$
$$\lceil .\pi \rceil = .\pi \qquad \lceil @p \rceil = @\lceil p \rceil$$

Clause typing $\boxed{f : T \vdash_\Sigma cl}$

$$\frac{PV(\bar{q}) = dom(\Gamma) \qquad \Gamma \mid f : T \vdash_\Sigma \lceil \bar{q} \rceil : U \qquad \Gamma \vdash_\Sigma t : U}{f : T \vdash_\Sigma (\Gamma \rhd f\ \bar{q} : U = t)}$$

$$\frac{PV(\bar{q}) = dom(\Gamma) \qquad \Gamma \mid f : T \vdash_\Sigma \lceil \bar{q} \rceil : U \qquad \forall i.\ \Gamma \vdash_\Sigma \Gamma(x_i) \text{ empty}}{f : T \vdash_\Sigma (\Gamma \rhd f\ \bar{q} : U = \text{absurd}\,\vec{x})}$$

Constructor typing $\boxed{\Delta \mid U : \Delta' \to s \vdash_\Sigma c : T}$

$$\frac{\Delta \vdash_\Sigma \Gamma \to T : s \qquad \Delta, \Gamma \vdash_\Sigma \bar{v} : \Delta' \qquad \Delta, \Gamma \vdash_\Sigma T = U\ \bar{v} : s}{\Delta \mid U : \Delta' \to s \vdash_\Sigma c : \Gamma \to T}$$

Declaration typing $\boxed{\Sigma \vdash d \rightsquigarrow \Sigma'}$

$$\frac{f \notin \Sigma \qquad \cdot \vdash_\Sigma T : s}{\Sigma \vdash f : T \rightsquigarrow \Sigma, function\ f : T\ where\ \cdot}$$

$$\frac{\forall i.\ f : T \vdash_\Sigma cl_i}{\Sigma[function\ f : T\ where\ \cdot] \vdash \overline{cl} \rightsquigarrow \Sigma[function\ f : T\ where\ \overline{cl}]}$$

$$\frac{D \notin \Sigma \qquad \vdash_\Sigma \Delta \qquad \Delta \vdash_\Sigma \Delta'}{\Sigma \vdash data\ D\ \Delta : \Delta' \to s \rightsquigarrow \Sigma, data\ D\ \Delta : \Delta' \to s}$$

$$\frac{\forall i.\ \Delta \mid D\ \Delta : \Delta' \to s \vdash c_i : T_i}{\Sigma[data\ D; \Delta : \Delta' \to s] \vdash_\Sigma data\ D\ \Delta\ where\ \overline{c : T} \rightsquigarrow \Sigma[data\ D\ \Delta : \Delta' \to s\ where\ \overline{c : T}]}$$

$$\frac{R \notin \Sigma \qquad \vdash_\Sigma \Delta}{\Sigma \vdash_\Sigma record\ R\ \Delta : s \rightsquigarrow \Sigma, record\ R\ \Delta : s}$$

$$\frac{\Delta \vdash_\Sigma \overline{(\hat{\pi} : T)} \to R\ \Delta : s \qquad U_i = \Delta \to (x : R\ \Delta) \to T_i[x.\pi_j / \hat{\pi}_j]}{\Sigma[record\ R\ \Delta : s] \vdash_\Sigma record\ R\ \Delta\ where\ constructor\ c, \overline{field\ \hat{\pi} : T}}\ x \notin dom(\Delta)$$
$$\rightsquigarrow \Sigma[record\ R\ \Delta : s\ where\ c : \Delta \to \overline{(\pi : T)} \to R\ \Delta; projections\ \overline{\pi : U}]$$

4 EVALUATION

Matching $\boxed{\Sigma \vdash e/q = \sigma}$.

$$\frac{}{\Sigma \vdash v/\lfloor u \rfloor = \cdot} \qquad \frac{}{\Sigma \vdash v/x = [v/x]} \qquad \frac{\Sigma \vdash \overline{v}/\overline{\sigma} = \sigma}{\Sigma \vdash c\overline{v}/c\overline{p} = \sigma} \qquad \frac{v\ !\ \pi_i/p_i = \sigma}{\Sigma \vdash v/c_{\vec{\pi}}\overline{p} = \sigma} \qquad \frac{}{\Sigma \vdash .\pi/.\pi = \cdot}$$

$$\frac{\Sigma \vdash v \to v' \qquad \Sigma \vdash v'/c\overline{p} = \sigma}{\Sigma \vdash v/c\overline{p} = \sigma}$$

Weak head reduction $\boxed{\Sigma \vdash t \to t'}$

$$\frac{f\overline{q} = t \in \Sigma \qquad \Sigma \vdash \overline{e}/\overline{q} = \sigma}{\Sigma \vdash f\overline{e}\overline{e}' \to t[\sigma]\ !\ \overline{e}'} \qquad \frac{}{\Sigma \vdash \cdot/\cdot = \cdot} \qquad \frac{\Sigma \vdash e/q = \sigma \qquad \Sigma \vdash \overline{e}/\overline{q} = \sigma'}{\Sigma \vdash e, \overline{e}/q, \overline{q} = \sigma \uplus \sigma'}$$

5 COVERAGE

Pattern refinement $\boxed{\Gamma \vdash \vec{p} : \Delta \stackrel{x \mapsto c}{\Longrightarrow} \Gamma' \vdash \vec{p'} : \Delta}$.

$$\frac{(x : D\,\vec{u}\,\vec{v} \in \Gamma) \qquad \Sigma \vdash D.c : T}{\Gamma \vdash \vec{p} : \Delta \stackrel{x \mapsto c}{\Longrightarrow} \Gamma' \vdash p\sigma[c\,\vec{y}/x] :?}$$

6 TERMINATION

7 POSITIVITY

8 EXTENSIONS

8.1 Extended record declarations

Record types

```
record R  : s where
  [constructor c]
  [(co)inductive]
  [(no-)eta-equality]
  rd*
```

Record declaration

```
rd ::= field  : T
     | ts
     | cl
```

ACKNOWLEDGMENTS