A Dynamic Dependent Type Theory with Type-in-Type and Recursion

February 25, 2021

1 type soundness or blame

The proof follows the standard structure:

- All judgments respect weakening and well typed substitution
- Most judgments are marked with types to make subject reduction obvious (assuming the substitution lemma)
- definitional equality is defined in terms par-reductions, which (via confluence)
  - gives transitivity to equality
  - means that type constructors are unique
- for preservation, function elimination is the only interesting case
  - the stack of casts is inspected, all casts are values (usually either $\star$ or Π)
    - if all casts are Π then coersions can be calculated and a reduction can step
    - if any casts are not Π there is a specific source location and observation to blame

1.1 Structural Properties

1.1.1 Weakening

For any derivation of $H \vdash A : \star$ the following rules are admissible:

$$\frac{H, H' \vdash}{H, x : A, H' \vdash}$$

$$\frac{H, H' \vdash b : B}{H, x : A, H' \vdash b : B}$$

$$\frac{H, H' \vdash e : \overline{\star}}{H, x : A, H' \vdash e : \overline{\star}}$$

$$\frac{H, H' \vdash b \equiv b' : B}{H, x : A, H' \vdash b \equiv b' : B}$$

$$\frac{H, H' \vdash b \Rrightarrow_* b' : B}{H, x : A, H' \vdash b \Rrightarrow_* b' : B}$$

$$\frac{H, H' \vdash b \Rrightarrow b' : B}{H, x : A, H' \vdash b \Rrightarrow b' : B}$$

$$\frac{H, H' \vdash e \Rrightarrow e' : \overline{\star}}{H, x : A, H' \vdash e \Rrightarrow e' : \overline{\star}}$$

$$\frac{H, H' \vdash o \Rrightarrow o'}{H, x : A, H' \vdash o \Rrightarrow o'}$$

1.2 Substitution

1.2.1 $\Rrightarrow$ is reflexive

The following rules are admissible:

$$\frac{H \vdash a : A}{H \vdash a \Rrightarrow\ a : A}$$

$$\frac{H \vdash e : \overline{\star}}{H \vdash e \Rrightarrow e' : \overline{\star}}$$

$$\frac{H \vdash}{H \vdash o \Rrightarrow o}$$

by mutual induction

1.2.2 $\sim$ is reflexive

The following rule is admissible:

$$\frac{}{a \sim a}\sim\text{-refl}$$

by induction

1.2.3 $\equiv$ is reflexive

The following rule is admissible:

$$\frac{H \vdash a : A}{H \vdash a \equiv a : A}\equiv\text{-refl}$$

by $\Rrightarrow *$-refland $\sim$-refl

1.2.4 Context substitution

For any derivation of $H \vdash a : A$ the following rules are admissible:

$$\frac{H, x : A, H' \vdash}{H, H'\left[x \coloneqq A\right] \vdash}$$

$$\frac{H, x : A, H' \vdash b : B}{H, H'\left[x \coloneqq a\right] \vdash b\left[x \coloneqq a\right] \, : \, B\left[x \coloneqq a\right]}$$

$$\frac{H, x : A, H' \vdash e : \overline{\star}}{H, H'\left[x \coloneqq a\right] \vdash e\left[x \coloneqq a\right] \, : \overline{\star}}$$

$$\frac{H, x : A, H' \vdash e : \overline{\star}}{H, H'\left[x \coloneqq a\right] \vdash e\left[x \coloneqq a\right] \, : \overline{\star}}$$

$$\frac{H, x : A, H' \vdash b \equiv b' : B}{H, H'\left[x \coloneqq a\right] \vdash b\left[x \coloneqq a\right] \equiv b'\left[x \coloneqq a\right] : B\left[x \coloneqq a\right]}$$

$$\frac{H, x : A, H' \vdash b \Rrightarrow_{*} b' : B}{H, H'\left[x \coloneqq a\right] \vdash b\left[x \coloneqq a\right] \Rrightarrow_{*} b'\left[x \coloneqq a\right] : B\left[x \coloneqq a\right]}$$

$$\frac{H, x : A, H' \vdash b \Rrightarrow b' : B}{H, H'\left[x \coloneqq a\right] \vdash b\left[x \coloneqq a\right] \Rrightarrow b'\left[x \coloneqq a\right] : B\left[x \coloneqq a\right]}$$

$$\frac{H, x : A, H' \vdash e \Rrightarrow e' : \overline{\star}}{H, H'\left[x \coloneqq a\right] \vdash e\left[x \coloneqq a\right] \Rrightarrow e'\left[x \coloneqq a\right] \, : \overline{\star}}$$

$$\frac{H, x : A, H' \vdash o \Rrightarrow o'}{H, H'\left[x \coloneqq a\right] \vdash o\left[x \coloneqq a\right] \Rrightarrow o'\left[x \coloneqq a\right]}$$

$$\frac{H, x : A, H' \vdash b \sim b' : B}{H, H'\left[x \coloneqq a\right] \vdash b\left[x \coloneqq a\right] \sim b'\left[x \coloneqq a\right] : B\left[x \coloneqq a\right]}$$

$$\frac{H, x : A, H' \vdash e \sim e' : \overline{\star}}{H, H'\left[x \coloneqq a\right] \vdash e\left[x \coloneqq a\right] \sim e'\left[x \coloneqq a\right] : \overline{\star}}$$

$$\frac{H, x : A, H' \vdash e \, Elim_{\Pi} y : e_A . e_B}{H, H'\left[x \coloneqq a\right] \vdash e\left[x \coloneqq a\right] \, Elim_{\Pi} y : e_A\left[x \coloneqq a\right] . e_B\left[x \coloneqq a\right]}$$

$$\frac{H, x : A, H' \vdash e \, Elim_{\star}}{H, H'\left[x \coloneqq a\right] \vdash e\left[x \coloneqq a\right] \, Elim_{\star}}$$

by mutual induction on the derivations with reflexivity lemmas.

1.3 Computation

1.3.1 $\Rrightarrow$ $Elim_{\star}$

if $e \, Elim_{\star}$ and $e \Rrightarrow e'$ then $e' \, Elim_{\star}$
by induction on $Elim_{\star}$

1.3.2 $\Rrightarrow$-substitution

For any $a \Rrightarrow a'$. The following rules are admissible:

$$\frac{b \Rrightarrow b'}{b\,[x \coloneqq a] \Rrightarrow b'\,[x \coloneqq a']}$$

$$\frac{e \Rrightarrow e}{e\,[x \coloneqq a] \Rrightarrow e'\,[x \coloneqq a']}$$

$$\frac{e \Rrightarrow e' \quad e\,[x \coloneqq a]\ \mathit{Elim}_{\Pi} x : e_A\,[x \coloneqq a]\,.e_B\,[x \coloneqq a]}{e_A \Rrightarrow e'_A \quad e_B \Rrightarrow e'_B \quad e'\,[x \coloneqq a']\ \mathit{Elim}_{\Pi} x : e'_A\,[x \coloneqq a']\,.e'_B\,[x \coloneqq a']}$$

by mutual induction on the derivations

1.3.3 $\Rrightarrow$ preserves type in destination

$$\frac{H \vdash a \Rrightarrow a' : A}{H \vdash a' : A}$$

Since the apparent type of a will at most $A \Rrightarrow A'$ (by $\Rrightarrow$-substitution) so $H \vdash A \equiv A' : \star$, and follows from conversion

1.3.4 $\Rrightarrow_*$ preserves type

The following rule is admissible:

$$\frac{H \vdash a \Rrightarrow_* a' : A}{H \vdash a : A}$$

by induction

$$\frac{H \vdash a \Rrightarrow_* a' : A}{H \vdash a' : A}$$

by induction

1.3.5 $\sim$ preserves type

The following rules are admissible:

$$\frac{H \vdash a \sim a' : A}{H \vdash a' : A}$$

by induction

1.3.6 $\equiv$ preserves type

The following rules are admissible:

$$\frac{H \vdash a \equiv a' : A}{H \vdash a : A}$$

$$\frac{H \vdash a \equiv a' : A}{H \vdash a' : A}$$

by the def of $\Rrightarrow_*$

1.3.7 def of $-^*$

there is a maximal par-reduction step that can be computed for every syntactic form defined:

$$\begin{array}{rll}
\star^* & = \star & a^* \to a \\
(\Pi x : A.B)^* & = \Pi x : A^*.B^* & \\
(a_h :: e)^* & = a_h^* :: e^* & \\
((\mathsf{fun}\, f.\, y.b) :: e\, a)^* & = (b^* \left[f \coloneqq (\mathsf{fun}\, f.\, x.b^*), x \coloneqq a^* :: e_A^*\right] :: e_B^* \left[x \coloneqq a^*\right]) \text{ if } e\, Elim_\Pi x : e_A.e_B & a_h^* \to a \\
(b\, a)^* & = b^*\, a^* \text{ otherwise} & \\
x^* & = x & \\
(\mathsf{fun}\, f.\, x.b)^* & = \mathsf{fun}\, f.\, x.b^* & \\
(e =_{l,o} A)^* & = e^* =_{l,o^*} A^* & e^* \to e \\
.^* & = . & o^* \to o \\
(o.arg)^* & = o^*.arg & \\
(o.bod[b])^* & = o^*.bod[b^*] &
\end{array}$$

1.3.8 $-^*$ $Elim_\star$

if $e\, Elim_\star$ then $e^*\, Elim_\star$

by induction on $Elim_\star$

1.3.9 $-^*$ $Elim_\Pi$

if $e\, Elim_\Pi x : e_A.e_B$ then $e^*\, Elim_\Pi x : e_A^*.e_B^*$

by induction on $Elim_\Pi$

1.3.10 $-^*$ is maximal

- if $a \Rrightarrow a'$ then $a' \Rrightarrow a^*$
- if $e \Rrightarrow e'$ then $e' \Rrightarrow e^*$
- if $o \Rrightarrow o'$ then $o' \Rrightarrow o^*$

by mutual induction on $\Rrightarrow$ relations, interesting cases include

- $\Pi C \Rrightarrow$ since if $e\, Elim_\Pi x : e_A.e_B$ then $e^*\, Elim_\Pi x : e_A^*.e_B^*$
- $\Pi E \Rrightarrow$, $b\, a \Rrightarrow b'\, a'$

- if the elimination is not possible with b, follows from induction
- if the elimination is possible with b, it will still be possible with b' since, by induction $b \Rrightarrow b'$

1.3.11 $\Rrightarrow$ is confluent

if $H \vdash a \Rrightarrow b : A$ and $H \vdash a \Rrightarrow b' : A$ then there exists c such that
$H \vdash b \Rrightarrow c : A$ and $\Gamma \vdash b' \Rrightarrow c : A$
by the maximality of $-^*$

1.3.12 $\Rrightarrow_*$ is transitive

The following rule is admissible:

$$\frac{H \vdash a \Rrightarrow_* b : A \quad H \vdash b \Rrightarrow_* c : A}{H \vdash a \Rrightarrow_* c : A} \Rrightarrow *\text{-trans}$$

by induction

1.3.13 $\Rrightarrow_*$ is confluent

if $H \vdash a \Rrightarrow_* b : A$ and $H \vdash a \Rrightarrow_* b' : A$ then there exists c such that
$H \vdash b \Rrightarrow_* c : A$ and $H \vdash b' \Rrightarrow_* c : A$
Follows from $\Rrightarrow *$-trans and the confluence of $\Rrightarrow$ using standard techniques

1.3.14 $\sim$Equivalence

The following rules are admissible:

$$\frac{}{a \sim a'}$$

$$\frac{a \sim a'}{a' \sim a}$$

$$\frac{a \sim a' \quad a' \sim a''}{a' \sim a''}$$

each by induction

1.3.15 $\sim$ commutes with $\Rrightarrow$, $\Rrightarrow_*$

The following rules are admissible:

$$\frac{a \Rrightarrow a' \quad a \sim b}{b \Rrightarrow b' \quad a' \sim b'}$$

$$\frac{H \vdash a \Rrightarrow_* a' : A \quad a \sim b}{H \vdash b \Rrightarrow_* b' : A \quad a' \sim b'}$$

both by induction (observations can be ignored since $\Rrightarrow$ is reflexive)

1.3.16 $\equiv$ is symmetric

The following rule is admissible:

$$\frac{H \vdash a \equiv a' : A}{H \vdash a' \equiv a : A} \equiv\text{-sym}$$

by $\sim$Equivalence

1.3.17 $\equiv$ is transitive

$$\frac{H \vdash a \equiv b : A \qquad H \vdash b \equiv c : A}{H \vdash a \equiv c : A} \equiv\text{-trans}$$

by the confluence of $\Rrightarrow_*$and $\sim$ commutativity

1.3.18 $\rightsquigarrow$ preserves type

For any derivations of $H \vdash a : A$, $a \rightsquigarrow a'$,

$$H \vdash a' : A$$

since $\rightsquigarrow$ implies $\Rrightarrow$ and $\Rrightarrow$ preserves types

1.4 Type constructors

1.4.1 Type constructors are stable over $\Rrightarrow$

- if $* \Rrightarrow A$ then A is $*$
- if $* :: e \Rrightarrow A_h :: e'$ then A_h is $*$
- if $\Pi x : A.B \Rrightarrow C$ then C is $\Pi x : A'.B'$ for some A', B'
- if $\Pi x : A.B :: e \Rrightarrow C_h :: e'$ then C_h is $\Pi x : A'.B'$ for some A', B'

by induction on $\Rrightarrow$

1.4.2 Type constructors are stable over $\Rrightarrow_*$

- if $H \vdash * \Rrightarrow_* A : B$ then A_h is $*$
- if $H \vdash * :: e \Rrightarrow_* A_h :: e' : B$ then A_h is $*$
- if $H \vdash \Pi x : A.B \Rrightarrow_* C : D$ then C is $\Pi x : A'.B'$ for some A', B'
- if $H \vdash \Pi x : A.B :: e \Rrightarrow_* C_h :: e' : D$ then C_h is $\Pi x : A'.B'$ for some A', B'

by induction on $\Rrightarrow_*$ and

1.4.3 Type constructors are stable over $\sim$

- if $* \sim A$ then A is $*$
- if $* :: e \sim A_h :: e'$ then A_h is $*$
- if $\Pi x : A.B \sim C$ then C is $\Pi x : A'.B'$ for some A', B'
- if $\Pi x : A.B :: e \sim C_h :: e'$ then C_h is $\Pi x : A'.B'$ for some A', B'

by induction on $\sim$

1.4.4 Type constructors definitionaly unique

for any H, A, B, C, e, e'

- $H \vdash * \not\equiv \Pi x : A.B \ : C$
- $H \vdash * :: e \not\equiv \Pi x : A.B \ : C$
- $H \vdash * \not\equiv \Pi x : A.B :: e \ : C$
- $H \vdash * :: e \not\equiv \Pi x : A.B :: e' \ : C$

from constructor stability

1.5 Canonical forms

If $\lozenge \vdash v_h \ : \Pi x : A.B$, then v_h is $\mathsf{fun}\, f.\, x.b$, since it is the only applicable rule

1.6 Type simplification

To minimize bookkeeping, when $\lozenge \vdash v_{eq} : \overline{\star}$

- $\star :: v_{eq}$ can be said to simplify to $\star$ if each v_{eq} simplifies to $\star$ (if it does not simplify there is a source of blame)
- $\Pi x : A.B :: v_{eq}$ can be said to simplify to $\Pi x : A.B$ if each v_{eq} simplifies to $\star$ (if it does not simplify there is a source of blame)

1.7 Progress

$\lozenge \vdash c : A$ implies that c is a value, there exists c' such that $c \rightsquigarrow c'$, or a static location can be blamed. and $\lozenge \vdash e : \overline{\star}$ implies that e is a value, there exists e' such that $e \rightsquigarrow e'$, or a static location can be blamed

By mutual induction on the typing derivations with the help of the canonical forms lemma

Explicitly:

cast typing

- $eq - ty - 1$ by **induction**

- $eq - ty - 2$ by **induction**

term typing

- c is typed by type-in-type. c is $\star$, a value
- c is typed by $\Pi - ty$. a is a value
- c is typed by the conversion rule, then by **induction**
- c is typed by the *apparent* rule, then c is $a_h :: e$ by each head typing. By induction e is a value, there exists e' such that $e \rightsquigarrow e'$. If there is blame that blame can be used, if $e \rightsquigarrow e'$ preform the step. otherwise e is a value:
  - a_h cannot be typed by the variable rule in the empty context
  - a_h is typed by type-in-type. a is $\star$.
  - a_h is typed by $\Pi - ty$. a is a value
  - a_h is typed by $\Pi - \mathsf{fun} - ty$. a is a value
  - a_h is typed by $\Pi - app - ty$. Then a_h is $b\, a$, and there are derivations of $\Diamond \vdash b : \Pi x : A.B$, and $\Diamond \vdash a : A$ for some A and B. By **induction** a is a value, there exists a' such that $a \rightsquigarrow a'$, or blame and b is a value or there exists b' such that $b \rightsquigarrow b'$ or blame.
    - if b and a are values, then b is $b_h :: v_{eq}$, where $v_{eq} \uparrow$ is $\Pi x : A_{\uparrow}.B_{\uparrow}$ (or $v_{eq} \uparrow$ is $\Pi x : A_{\uparrow}.B_{\uparrow} :: e$, and by simplification $\Pi x : A_{\uparrow}.B_{\uparrow}$ or blame can be produced) (by **stability**)
      - if $v_{eq}\, Elim_{\Pi}\, x : e_A.e_B$ then $v_{eq} \downarrow$ is $\Pi x : A_{\downarrow}.B_{\downarrow}$ (or $\Pi x : A_{\downarrow}.B_{\downarrow} :: e$, and by simplification $\Pi x : A_{\downarrow}.B_{\downarrow}$ or blame can be produced) by **Canonical forms** b_h is $(\mathsf{fun}\, f.\, x.b')$ and the step is $((\mathsf{fun}\, f.\, x.b) :: v_{eq}\, v) :: v'_{eq} \rightsquigarrow (b\,[f \coloneqq (\mathsf{fun}\, f.\, x.b)\,, x \coloneqq v :: e_A] :: e'_B\,[x \coloneqq v])$ (implicitly uses that $Elim_{\Pi}$is deterministic in its first argument)
      - if $v_{eq}\, \cancel{Elim_{\Pi}}$ then there must exist $\left[\cancel{\Pi} =_{l,o} \Pi x : A''.B''\right] \in v_{eq}$ (with simplification) and l, o can be blamed
    - if b or a can construct blame then $b\, a$ can use that blame
    - if b is a value and $a \rightsquigarrow a'$ then $b\, a \rightsquigarrow b\, a'$
    - if $b \rightsquigarrow b'$ then $b\, a \rightsquigarrow b'\, a$

1.8 Type Soundness

For any well typed term in an empty context, no sequence of small step reductions will cause a computation to “get stuck” without blame. Either a final value will be reached, further reductions can be taken, or blame is omitted. This follows by iterating the progress and preservation lemmas.

2 Elaboration Embeds Typing

$\vdash m : M$, $\vdash M\, Elab_{\star,l}\, A$, and $\vdash m\, Elab_{A,l}\, a$ then $\vdash a : A$

Sketch (the Surface type system has slight but pervasive changes to the language described in the "baselanguage" folder),

- strengthen the hypothesis to $\Gamma\, Elab\, H$, $\Gamma \vdash m : M$, $H \vdash M\, Elab_{\star,l}\, A$, and $H \vdash m\, Elab_{A,l}\, a$ then $H \vdash a : A$
- follows by mutual induction

3 Computation resulting in blame cannot be typed in the surface language

$\vdash a : A$ and a blame then there is no $\vdash m : M$ such that $\vdash M\, Elab_{\star,l}\, A$, and $\vdash m\, Elab_{A,l'}\, a$

Sketch: if $\vdash m : M$ then $\vdash a : A$ are elaborated without source labels (l, l' are superfluous) therefore blame is impossible to construct

4 Computation in the cast language respects computation in the surface language

$\vdash A : *$ and $\vdash M\, Elab_{\star,l}\, A$ then

1. if $A \rightsquigarrow_* *$ then $M \rightsquigarrow_* *$
2. if $A \rightsquigarrow_* \Pi x : B.C$ then $M \rightsquigarrow_* \Pi x : N.P$

Sketch: evaluation is designed to be "correct by construction" . Casts and cast evaluation steps can be completely removed, resulting in exactly the small steps of the surface language