

Type Soundness in an Intensional Dependent Type Theory with Type-in-Type and Recursion

February 23, 2021

1 Examples

logical unsoundness:

$$\text{fun } f : (x.x) . x : \star . f x \quad : \Pi x : \star . x$$

1.1 some constructs

While logically unsound, the language is extremely expressive. The following calculus of Constructions constructs are expressible,

$$\begin{aligned} a_1 =_A a_2 &:= \lambda A : \star . \lambda a_1 : A . \lambda a_2 : A . \Pi C : (A \rightarrow \star) . C a_1 \rightarrow C a_2 \\ \text{refl}_{a:A} &:= \lambda C : (A \rightarrow \star) . \lambda x : C a . x \quad : a =_A a \\ \text{Unit} &:= \Pi A : \star . A \rightarrow A \\ tt &:= \lambda A : \star . \lambda a : A . a \\ \perp &:= \Pi x : \star . x \\ \neg A &:= \Pi A : \star . A \rightarrow \perp \end{aligned}$$

1.1.1 Church Booleans

$$\begin{aligned} \mathbb{B}_c &:= \Pi A : \star . A \rightarrow A \rightarrow A \\ \text{true}_c &:= \lambda A : \star . \lambda \text{then} : A . \lambda \text{else} : A . \text{then} \\ \text{false}_c &:= \lambda A : \star . \lambda \text{then} : A . \lambda \text{else} : A . \text{else} \end{aligned}$$

1.1.2 Church \mathbb{N}

$$\begin{aligned} \mathbb{N}_c &:= \Pi A : \star . (A \rightarrow A) \rightarrow A \rightarrow A \\ 0_c &:= \lambda A : \star . \lambda s : (A \rightarrow A) . \lambda z : A . z \\ 1_c &:= \lambda A : \star . \lambda s : (A \rightarrow A) . \lambda z : A . s z \\ 2_c &:= \lambda A : \star . \lambda s : (A \rightarrow A) . \lambda z : A . s (s z) \\ &\dots \end{aligned}$$

1.1.3 Large Elimination

Since there is type-in-type, a kind of large elimination is possible

$\lambda b : \mathbb{B}_c.b \star Unit \perp$
 $\lambda n : \mathbb{N}_c.n \star (\lambda - .Unit) \perp$
 thus $\neg 1_c =_{\mathbb{N}_c} 0_c$ is provable (in a non trivial way):
 $\lambda pr : (HC : (\mathbb{N}_c \rightarrow \star).C 1_c \rightarrow C 0_c).pr (\lambda n : \mathbb{N}_c.n \star (\lambda - .Unit) \perp) tt$:
 $\neg 1_c =_{\mathbb{N}_c} 0_c$
 $\neg true_c =_{\mathbb{B}_c} false_c$ is provable:
 $\lambda pr : (HC : (\mathbb{B}_c \rightarrow \star).C true_c \rightarrow C false_c).pr (\lambda b : \mathbb{B}_c.b \star Unit \perp) tt$:
 $\neg true_c =_{\mathbb{B}_c} false_c$
 $\neg Unit =_{\star} \perp$ is provable:
 $\lambda pr : (HC : (\star \rightarrow \star).C Unit \rightarrow C \perp).pr (\lambda x.x) tt$: $\neg Unit =_{\star} \perp$
 $\neg \star =_{\star} \perp$ is provable:
 $\lambda pr : (HC : (\star \rightarrow \star).C \star \rightarrow C \perp).pr (\lambda x.x) \perp$: $\neg \star =_{\star} \perp$
 There are more examples in [1] where Cardelli has studied a similar system.

2 Properties

2.1 Contexts

2.1.1 Sub-Contexts are well formed

The following rules are admissible:

$$\begin{array}{c}
 \frac{\Gamma, \Gamma' \vdash}{\Gamma \vdash} \\
 \\
 \frac{\Gamma, \Gamma' \vdash M : \sigma}{\Gamma \vdash} \\
 \\
 \frac{\Gamma, \Gamma' \vdash M \Rightarrow M' : \sigma}{\Gamma \vdash} \\
 \\
 \frac{\Gamma, \Gamma' \vdash M \Rightarrow_{\star} M' : \sigma}{\Gamma \vdash} \\
 \\
 \frac{\Gamma, \Gamma' \vdash M \equiv M' : \sigma}{\Gamma \vdash}
 \end{array}$$

by mutual induction on the derivations.

2.1.2 Context weakening

For any derivation of $\Gamma \vdash \sigma : \star$, the following rules are admissible:

$$\frac{\Gamma, \Gamma' \vdash}{\Gamma, x : \sigma, \Gamma' \vdash}$$

$$\begin{array}{c}
\frac{\Gamma, \Gamma' \vdash M : \tau}{\Gamma, x : \sigma, \Gamma' \vdash M : \tau} \\
\\
\frac{\Gamma, \Gamma' \vdash M \Rightarrow M' : \sigma}{\Gamma, x : \sigma, \Gamma' \vdash M \Rightarrow M' : \sigma} \\
\\
\frac{\Gamma, \Gamma' \vdash M \Rightarrow_* M' : \sigma}{\Gamma, x : \sigma, \Gamma' \vdash M \Rightarrow_* M' : \sigma} \\
\\
\frac{\Gamma, \Gamma' \vdash M \equiv M' : \tau}{\Gamma, x : \sigma, \Gamma' \vdash M \equiv M' : \tau}
\end{array}$$

by mutual induction on the derivations.

2.1.3 \Rightarrow is reflexive

The following rule is admissible:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M \Rightarrow M : \sigma} \Rightarrow\text{-refl}$$

by induction

2.1.4 \equiv is reflexive

The following rule is admissible:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M \equiv M : \sigma} \equiv\text{-refl}$$

by \Rightarrow *-refl

2.1.5 Context substitution

For any derivation of $\Gamma \vdash N : \tau$ the following rules are admissible:

$$\begin{array}{c}
\frac{\Gamma, x : \tau, \Gamma' \vdash}{\Gamma, \Gamma' [x := N] \vdash} \\
\\
\frac{\Gamma, x : \tau, \Gamma' \vdash M : \sigma}{\Gamma, \Gamma' [x := N] \vdash M [x := N] : \sigma [x := N]} \\
\\
\frac{\Gamma, x : \tau, \Gamma' \vdash M \Rightarrow M' : \sigma}{\Gamma, \Gamma' [x := N] \vdash M [x := N] \Rightarrow M' [x := N] : \sigma [x := N]} \\
\\
\frac{\Gamma, x : \tau, \Gamma' \vdash M \Rightarrow_* M' : \sigma}{\Gamma, \Gamma' [x := N] \vdash M [x := N] \Rightarrow_* M' [x := N] : \sigma} \\
\\
\frac{\Gamma, x : \tau, \Gamma' \vdash M \equiv M' : \sigma}{\Gamma, \Gamma' [x := N] \vdash M [x := N] \equiv M' [x := N] : \sigma [x := N]}
\end{array}$$

by mutual induction on the derivations. Specifically, at every usage of x from the var rule in the original derivation, replace the usage of the var rule with the derivation of $\Gamma \vdash N : \tau$ weakened to the context of $\Gamma, \Gamma' [x := N] \vdash N : \tau$, and apply \Rightarrow -refl, \Rightarrow_* -refl or \equiv -refl when needed.

2.2 Computation

2.2.1 \Rightarrow preserves type of source

The following rule is admissible:

$$\frac{\Gamma \vdash N \Rightarrow N' : \tau}{\Gamma \vdash N : \tau}$$

by induction

2.2.2 \Rightarrow -substitution

The following rule is admissible:

$$\frac{\Gamma, x : \sigma, \Gamma' \vdash M \Rightarrow M' : \tau \quad \Gamma \vdash N \Rightarrow N' : \sigma}{\Gamma, \Gamma' [x := N] \vdash M [x := N] \Rightarrow M' [x := N'] : \tau [x := N]}$$

by induction on the \Rightarrow derivations

2.2.3 \Rightarrow is confluent

if $\Gamma \vdash M \Rightarrow N : \sigma$ and $\Gamma \vdash M \Rightarrow N' : \sigma$ then there exists P such that

$$\Gamma \vdash N \Rightarrow P : \sigma \text{ and } \Gamma \vdash N' \Rightarrow P : \sigma$$

by standard techniques

2.3 \Rightarrow_*

2.3.1 \Rightarrow_* is transitive

The following rule is admissible:

$$\frac{\Gamma \vdash M \Rightarrow_* M' : \sigma \quad \Gamma \vdash M' \Rightarrow_* M'' : \sigma}{\Gamma \vdash M \Rightarrow_* M'' : \sigma} \Rightarrow_*\text{-trans}$$

by induction

2.3.2 \Rightarrow preserves type in destination

$$\frac{\Gamma \vdash N \Rightarrow N' : \tau}{\Gamma \vdash N' : \tau}$$

By induction on the \Rightarrow derivation with the help of the substitution lemma.

- $\Pi \Rightarrow$

- $M' [x := N', f := (\text{fun } f : (x.\tau') . x : \sigma'.M')] : \tau' [x := N']$ by the substitution lemma used on the inductive hypotheses
- $\tau [x := N] \Rightarrow \tau' [x := N']$ by \Rightarrow -substitution, so $\tau [x := N] \equiv \tau' [x := N']$
- by the conversion rule $M' [x := N', f := (\text{fun } f : (x.\tau') . x : \sigma'.M')] : \tau [x := N]$
- $\Pi\text{-E} \Rightarrow$
 - $M' N' : \tau [x := N']$, by \Rightarrow -substitution and reflexivity, $\tau [x := N] \Rightarrow \tau [x := N']$, so $\tau [x := N] \equiv \tau [x := N']$
 - by the conversion rule $M' N' : \tau [x := N]$
- $\Pi\text{-I} \Rightarrow$
 - $\text{fun } f : (x.\tau') . x : \sigma'.M' : \Pi x : \sigma'.\tau', \Pi x : \sigma.\tau \Rightarrow \Pi x : \sigma'.\tau'$, so $\Pi x : \sigma.\tau \equiv \Pi x : \sigma'.\tau'$
 - by the conversion rule $\text{fun } f : (x.\tau') . x : \sigma'.M' : \Pi x : \sigma.\tau$
- all other cases are trivial

2.3.3 \Rightarrow_* preserves type

The following rule is admissible:

$$\frac{\Gamma \vdash M \Rightarrow_* M' : \sigma}{\Gamma \vdash M : \sigma}$$

by induction

$$\frac{\Gamma \vdash M \Rightarrow_* M' : \sigma}{\Gamma \vdash M' : \sigma}$$

by induction

2.3.4 \Rightarrow_* is confluent

if $\Gamma \vdash M \Rightarrow_* N : \sigma$ and $\Gamma \vdash M \Rightarrow_* N' : \sigma$ then there exists P such that

$$\Gamma \vdash N \Rightarrow_* P : \sigma \text{ and } \Gamma \vdash N' \Rightarrow_* P : \sigma$$

Follows from \Rightarrow *-trans and the confluence of \Rightarrow using standard techniques

2.3.5 \equiv is symmetric

The following rule is admissible:

$$\frac{\Gamma \vdash M \equiv N : \sigma}{\Gamma \vdash N \equiv M : \sigma} \equiv\text{-sym}$$

trivial

2.3.6 \equiv is transitive

$$\frac{\Gamma \vdash M \equiv N : \sigma \quad \Gamma \vdash N \equiv P : \sigma}{\Gamma \vdash M \equiv P : \sigma} \equiv\text{-trans}$$

by the confluence of \Rightarrow_*

2.3.7 \equiv preserves type

The following rules are admissible:

$$\frac{\Gamma \vdash M \equiv M' : \sigma}{\Gamma \vdash M : \sigma}$$

$$\frac{\Gamma \vdash M \equiv M' : \sigma}{\Gamma \vdash M' : \sigma}$$

by the def of \Rightarrow_*

2.3.8 Regularity

The following rule is admissible:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \sigma : \star}$$

by induction with \equiv -preservation for the Conv case

2.3.9 \rightsquigarrow implies \Rightarrow

For any derivations of $\Gamma \vdash M : \sigma$, $M \rightsquigarrow M'$

$$\Gamma \vdash M \Rightarrow M' : \sigma$$

by induction on \rightsquigarrow

2.3.10 \rightsquigarrow preserves type

For any derivations of $\Gamma \vdash M : \sigma$, $M \rightsquigarrow M'$,

$$\Gamma \vdash M' : \sigma$$

since \rightsquigarrow implies \Rightarrow and \Rightarrow preserves types

2.4 Type constructors

2.4.1 Type constructors are stable

- if $\Gamma \vdash * \Rightarrow M : \sigma$ then M is $*$
- if $\Gamma \vdash * \Rightarrow_* M : \sigma$ then M is $*$
- if $\Gamma \vdash \Pi x : \sigma. \tau \Rightarrow M : \sigma$ then M is $\Pi x : \sigma'. \tau'$ for some σ', τ'

- if $\Gamma \vdash \Pi x : \sigma.\tau \Rightarrow_* M : \sigma$ then M is $\Pi x : \sigma'.\tau'$ for some σ', τ'

by induction on the respective relations

2.4.2 Type constructors definitionally unique

There is no derivation of $\Gamma \vdash * \equiv \Pi x : \sigma.\tau : \sigma'$ for any $\Gamma, \sigma, \tau, \sigma'$
from \equiv -Def and constructor stability

2.5 Canonical forms

If $\Diamond \vdash v : \sigma$ then

- if σ is \star then v is \star or $\Pi x : \sigma.\tau$
- if σ is $\Pi x : \sigma'.\tau$ for some σ', τ then v is $\text{fun } f : (x.\tau') . x : \sigma''.P'$ for some τ', σ'', P'

By induction on the typing derivation

- Conv,
 - if σ is \star then eventually, it was typed with type-in-type, or Π -F. it could not have been typed by Π -I since constructors are definitionally unique
 - if σ is $\Pi x : \sigma'.\tau$ then eventually, it was typed with Π -I. it could not have been typed by type-in-type, or Π -F since constructors are definitionally unique
- type-in-type, $\Diamond \vdash v : \sigma$ is $\Diamond \vdash \star : \star$
- Π -F, $\Diamond \vdash v : \sigma$ is $\Diamond \vdash \Pi x : \sigma.\tau : \star$
- Π -I, $\Diamond \vdash v : \sigma$ is $\Diamond \vdash \text{fun } f : (x.\tau) . x : \sigma.M : \Pi x : \sigma.\tau$
- no other typing rules are applicable

2.6 Progress

$\Diamond \vdash M : \sigma$ implies that M is a value or there exists N such that $M \rightsquigarrow N$.

By direct induction on the typing derivation with the help of the canonical forms lemma

Explicitly:

- M is typed by the conversion rule, then by **induction**, M is a value or there exists N such that $M \rightsquigarrow N$
- M cannot be typed by the variable rule in the empty context
- M is typed by type-in-type. M is \star , a value

- M is typed by Π -F. M is $\Pi x : \sigma.\tau$, a value
- M is typed by Π -I. M is $\text{fun } f : (x.\tau).x : \sigma.M'$, a value
- M is typed by Π -E. M is $P N$ then exist some σ, τ for $\Diamond \vdash P : \Pi x : \sigma.\tau$ and $\Diamond \vdash N : \sigma$. By **induction** (on the P branch of the derivation) P is a value or there exists P' such that $P \rightsquigarrow P'$. By **induction** (on the N branch of the derivation) N is a value or there exists N' such that $N \rightsquigarrow N'$
 - if P is a value then by **canonical forms**, P is $\text{fun } f : (x.\tau).x : \sigma.P'$ and
 - * if N is a value then the one step reduction is $(\text{fun } f : (x.\tau).x : \sigma.P') N \rightsquigarrow P' [x := N, f := \text{fun } f : (x.\tau).x : \sigma.M]$
 - * otherwise there exists N' such that $N \rightsquigarrow N'$, and the one step reduction is $(\text{fun } f : (x.\tau).x : \sigma.P') N \rightsquigarrow (\text{fun } f : (x.\tau).x : \sigma.P') N'$
 - otherwise, there exists P' such that $P \rightsquigarrow P'$ and the one step reduction is $P N \rightsquigarrow P' N$

2.7 Type Soundness

For any well typed term in an empty context, no sequence of small step reductions will cause result in a computation to “get stuck”. Either a final value will be reached or further reductions can be taken. This follows by iterating the progress and preservation lemmas.

3 Non-Properties

- decidable type checking
- normalization/logical soundness

Definitional Equality does not preserve type constructors on the nose

If $\Gamma \vdash \sigma \equiv \sigma' : \star$ then

if σ is $\Pi x : \sigma''.\tau$ for some σ'', τ then σ' is $\Pi x : \sigma'''.\tau'$ for some σ''', τ'

counter example $\vdash \Pi x : \star.\star \equiv (\lambda x : \star.x)(\Pi x : \star.\star) : \star$

this implies the additional work in the Canonical forms lemma

4 differences from implementation

differences from Agda development

- in both presentations standard properties of variables binding and substitution are assumed

- In Agda the parallel reduction relation does not track the original typing judgment. This should not matter for the proof of confluence.
- only proved the function part of the canonical forms lemma (all that is needed for the proof)

differences from prototype

- bidirectional, type annotations are not always needed on functions
- toplevel recursion in addition to function recursion
- type annotations are not relevant for definitional equality

5 Proof improvements

- Clarify what unsoundness means
- proof outline at the top of document
- better function notation
- transition away from Greek letters

References

- [1] Luca Cardelli. *A Polymorphic λ -calculus with Type: Type*. Technical Report, DEC SRC, 130 Lytton Avenue, Palo Alto, CA 94301. May. SRC Research Report, 1986.