Strong normalization for simply-typed combinatory algebra using Girard's reducibility candidates formalized in Agda

Andreas Abel
Department of Computer Science, Gothenburg University, Sweden

November 1, 2020

This document provides a formal proof of strong normalization for combinatory algebra with the two combinators S and K. The result follow from a model construction where each type is interpreted as a reducibility candidate à la Girard. We thus demonstrate Girard's method in the most simple setting. In particular, since combinatory algebra is a variable-free language, we forgo the need to define substitution. The proof has been formalized in Agda 2.6.2 and this document reproduces the commented Agda code.

1 Preliminaries

We work in type theory with propositions-as types.

```
Proposition = Set
```

Negation: A proposition is false if it implies any other proposition.

```
¬̇_ : Proposition → Set₁
¬̇ A = ∀{C : Proposition} → A → C
```

2 Syntax

Types: For simplicity, we consider a single base type. Types are closed under function space formation.

```
infixr 6 _⇒_
```

```
data Ty : Set where
  o     : Ty
  _⇒_ : (a b : Ty) → Ty
```

We use small latin letters from the beginning of the alphabet to range over types.

```
variable a b c : Ty
```

Intrinsically well-typed terms of combinatory algebra (CA): these are applicative terms over the constants K and S.

```
infixl 5 _•_

data Tm : Ty → Set where
  K   : Tm (a ⇒ (b ⇒ a))
  S   : Tm ((c ⇒ (a ⇒ b)) ⇒ (c ⇒ a) ⇒ c ⇒ b)
  _•_ : (t : Tm (a ⇒ b)) (u : Tm a) → Tm b
```

We use small latin letters t, u and v to range over terms.

```
variable t t′ u u′ v v′ : Tm a
```

The reduction relation is given inductively via axioms for fully applied K and S and congruence rules for the reduction in either the function or the argument part of an application.

```
infix 4 _↦_

data _↦_ : (t t′ : Tm a) → Set where
  ↦K : K • t • u     ↦ t
  ↦S : S • t • u • v ↦ t • v • (u • v)
  ↦l : t ↦ t′   → t • u ↦ t′ • u
  f↦ : u ↦ u′ → t • u ↦ t • u′
```

3 Strong normalization

Sets of terms of a fixed type are expressed as predicates on terms of that type.

```
Pred : Ty → Set₁
Pred a = (t : Tm a) → Proposition

variable P Q : Pred a
```

The subset relation is implication of predicates.

```
infix 2 _⊂_

_⊂_ : (P Q : Pred a) → Proposition
P ⊂ Q = ∀{t} → P t → Q t
```

Strong normalization: a term is SN if all of its reducts are, inductively.

```
data SN (t : Tm a) : Proposition where
  acc : t ↦_ ⊂ SN → SN t
```

Reducts of SN terms are SN by definition.

```
sn-red : SN t → t ↦ t′ → SN t′
sn-red (acc sn) r = sn r
```

In combinatory algebra, the values are the underapplied functions. All values formed from SN components are SN. The proofs proceed by induction on the SN of the arguments, considering all possible one-step reducts of the values.

K is SN.

```
sn-K : SN (K {a} {b})
sn-K = acc λ()
```

K applied to one SN argument is SN.

```
sn-Kt : SN t → SN (K {a} {b} • t)
sn-Kt (acc snt) = acc λ{ (f↦ r) → sn-Kt (snt r) }
```

S is SN.

```
sn-S : SN (S {c} {a} {b})
sn-S = acc λ()
```

S applied to one SN argument is SN.

```
sn-St : SN t → SN (S • t)
sn-St (acc snt) = acc λ{ (f↦ r) → sn-St (snt r) }
```

S applied to two SN arguments is SN.

```
sn-Stu : SN t → SN u → SN (S • t • u)
sn-Stu (acc snt) (acc snu) = acc λ where
  (↦l (f↦ r)) → sn-Stu (snt r) (acc snu)
  (f↦ r)      → sn-Stu (acc snt) (snu r)
```

4 Reducibility candidates

Following Girard, terms which are not introductions are called neutral. In CA, the weak head redexes are the neutrals.

```
data Ne : Pred a where
  Ktu   : Ne (K • t • u)
  Stuv  : Ne (S • t • u • v)
  napp  : (n : Ne t) → Ne (t • u)
```

Partially applied combinators, i.e., values, are thus not neutral.

```
Kt¬ne : ¬̇ Ne (K {a} {b} • t)
Kt¬ne (napp ())

Stu¬ne : ¬̇ Ne (S • t • u)
Stu¬ne (napp (napp ()))
```

A reducibility candidate (CR) for a type is a set of SN terms of that type (condition CR1). Further, the set needs to be closed under reduction (CR2). Finally, a candidate needs to contain any neutral term of the right type whose reducts are already in the candidate.

```
record CR (P : Pred a) : Proposition where
  field
    cr1 : P ⊂ SN
    cr2 : P t → (t ↦_) ⊂ P
    cr3 : (n : Ne t) (h : t ↦_ ⊂ P) → P t
open CR
```

The set SN is a reducibility candidate.

```
sn−cr : CR (SN {a})
sn−cr .cr1 sn   = sn
sn−cr .cr2 sn   = sn−red sn
sn−cr .cr3 _ h = acc h
```

Given two reducibility candidates, one acting as the domain and one as the codomain, we form a new reducibility candidate, the function space.

The function space contains any SN term that, applied to a term in the domain, yields a result in the codomain.

```
record _⇨_ (P : Pred a) (Q : Pred b) (t : Tm (a ⇒ b)) : Proposition where
  field
    sn   : SN t
```

```
    app : ∀ {u} ((u) : P u) → Q (t • u)
  open _⇨_
```

The function space construction indeed operates on CRs.

CR1 holds by definition. The proof of CR2 only needs CR2 of the codomain. The proof of CR3 needs CR3 of the codomain and CR1 and CR2 of the domain.

```
  ⇨−cr : (crP : CR P) (crQ : CR Q) → CR (P ⇨ Q)
  ⇨−cr               crP crQ .cr1 (t)               = (t) .sn
  ⇨−cr               crP crQ .cr2 (t) r .sn         = sn−red ((t) .sn) r
  ⇨−cr               crP crQ .cr2 (t) r .app (u)    = crQ .cr2 ((t) .app (u)) (↦l r)
  ⇨−cr               crP crQ .cr3     n (t) .sn     = acc λ r → (t) r .sn
  ⇨−cr {P = P} {Q = Q} crP crQ .cr3 {t} n (t) .app (u) = loop (u) (crP .cr1 (u))
```

We perform a side induction on the SN of the function argument, exploiting that the domain is closed under reduction.

```
  where
  loop : ∀{u} → P u → SN u → Q (t • u)
  loop (u) (acc snu) = crQ .cr3 (napp n) λ where
    ↦K     → Kt¬ne n
    ↦S     → Stu¬ne n
    (↦l r) → (t) r .app (u)
    (f↦ r) → loop (crP .cr2 (u) r) (snu r)
```

5 Soundness

Interpretation of types as semantic types: we interpret the base type as the set of all SN terms of that type and the function type via the function space construction.

```
  ⟦_⟧ : ∀ a → Pred a
  ⟦ o ⟧      = SN
  ⟦ a ⇒ b ⟧ = ⟦ a ⟧ ⇨ ⟦ b ⟧
```

Types are indeed interpreted as CRs.

```
  ty−cr : ∀ a → CR ⟦ a ⟧
  ty−cr o        = sn−cr
  ty−cr (a ⇒ b) = ⇨−cr (ty−cr a) (ty−cr b)
```

Any term in a semantic type is SN.

$$\begin{aligned}&\mathsf{sem\text{-}sn} : [\![\, a \,]\!]\; t \to \mathsf{SN}\; t\\ &\mathsf{sem\text{-}sn}\; (\!| t |\!) = \mathsf{ty\text{-}cr}\; \_\; .\mathsf{cr1}\; (\!| t |\!)\end{aligned}$$

Interpretation of S: constant S, fully applied to terms inhabiting the respective semantic types, inhabits the correct semantic type as well.

This lemma is proven by induction on the SN of the subterms, redundant facts which we add explicitly for the sake of recursion. The induction hypothesis is applicable thanks to CR2.

$$\begin{aligned}(\!|\mathsf{S}|\!) :\;\; & [\![\, c \Rightarrow a \Rightarrow b \,]\!]\; t \to \mathsf{SN}\; t\\ \to\; & [\![\, c \Rightarrow a \,]\!]\; u \quad\;\; \to \mathsf{SN}\; u\\ \to\; & [\![\, c \,]\!]\; v \qquad\quad\;\; \to \mathsf{SN}\; v\\ \to\; & [\![\, b \,]\!]\; (\mathsf{S} \bullet t \bullet u \bullet v)\end{aligned}$$

$$\begin{aligned}&(\!|\mathsf{S}|\!)\; \{b = b\}\; (\!| t |\!)\; (\mathsf{acc}\; snt)\; (\!| u |\!)\; (\mathsf{acc}\; snu)\; (\!| v |\!)\; (\mathsf{acc}\; snv) = \mathsf{ty\text{-}cr}\; b\; .\mathsf{cr3}\; \mathsf{Stuv}\; \lambda\; \mathsf{where}\\ &\quad \mapsto\!\mathsf{S} \qquad\qquad\qquad\quad\; \to (\!| t |\!)\; .\mathsf{app}\; (\!| v |\!)\; .\mathsf{app}\; ((\!| u |\!)\; .\mathsf{app}\; (\!| v |\!))\\ &\quad (\mapsto\!\mathsf{l}\; (\mapsto\!\mathsf{l}\; (\mathsf{f}\!\mapsto rt))) \to (\!|\mathsf{S}|\!)\; (\mathsf{ty\text{-}cr}\; \_\; .\mathsf{cr2}\; (\!| t |\!)\; rt)\; (snt\; rt)\\ &\qquad\qquad\qquad\qquad\qquad\quad\; (\!| u |\!)\; (\mathsf{acc}\; snu)\\ &\qquad\qquad\qquad\qquad\qquad\quad\; (\!| v |\!)\; (\mathsf{acc}\; snv)\\ &\quad (\mapsto\!\mathsf{l}\; (\mathsf{f}\!\mapsto ru)) \qquad\; \to (\!|\mathsf{S}|\!)\; (\!| t |\!)\; (\mathsf{acc}\; snt)\\ &\qquad\qquad\qquad\qquad\qquad\quad\; (\mathsf{ty\text{-}cr}\; \_\; .\mathsf{cr2}\; (\!| u |\!)\; ru)\; (snu\; ru)\\ &\qquad\qquad\qquad\qquad\qquad\quad\; (\!| v |\!)\; (\mathsf{acc}\; snv)\\ &\quad (\mathsf{f}\!\mapsto rv) \qquad\qquad\quad\;\; \to (\!|\mathsf{S}|\!)\; (\!| t |\!)\; (\mathsf{acc}\; snt)\\ &\qquad\qquad\qquad\qquad\qquad\quad\; (\!| u |\!)\; (\mathsf{acc}\; snu)\\ &\qquad\qquad\qquad\qquad\qquad\quad\; (\mathsf{ty\text{-}cr}\; \_\; .\mathsf{cr2}\; (\!| v |\!)\; rv)\; (snv\; rv)\end{aligned}$$

Interpretation of K: analogously.

$$\begin{aligned}&(\!|\mathsf{K}|\!) : [\![\, a \,]\!]\; t \to \mathsf{SN}\; t \to \mathsf{SN}\; u \to [\![\, a \,]\!]\; (\mathsf{K} \bullet t \bullet u)\\ &(\!|\mathsf{K}|\!)\; \{a\}\; (\!| t |\!)\; (\mathsf{acc}\; snt)\; (\mathsf{acc}\; snu) = \mathsf{ty\text{-}cr}\; a\; .\mathsf{cr3}\; \mathsf{Ktu}\; \lambda\; \mathsf{where}\\ &\quad \mapsto\!\mathsf{K} \qquad\qquad\;\; \to (\!| t |\!)\\ &\quad (\mapsto\!\mathsf{l}\; (\mathsf{f}\!\mapsto rt)) \to (\!|\mathsf{K}|\!)\; (\mathsf{ty\text{-}cr}\; a\; .\mathsf{cr2}\; (\!| t |\!)\; rt)\; (snt\; rt)\; (\mathsf{acc}\; snu)\\ &\quad (\mathsf{f}\!\mapsto ru) \qquad\; \to (\!|\mathsf{K}|\!)\; (\!| t |\!)\; (\mathsf{acc}\; snt)\; (snu\; ru)\end{aligned}$$

Term interpretation: each term inhabits its respective semantic type.

Proof by induction on the term.

$$\begin{aligned}&(\!|\_|\!) : (t : \mathsf{Tm}\; a) \to [\![\, a \,]\!]\; t\\ &(\!|\; \mathsf{S}\; \{b = b\}\; |\!)\; .\mathsf{sn} &&= \mathsf{sn\text{-}S}\\ &(\!|\; \mathsf{S}\; \{b = b\}\; |\!)\; .\mathsf{app}\; (\!| t |\!)\; .\mathsf{sn} &&= \mathsf{sn\text{-}St}\; ((\!| t |\!)\; .\mathsf{sn})\\ &(\!|\; \mathsf{S}\; \{b = b\}\; |\!)\; .\mathsf{app}\; (\!| t |\!)\; .\mathsf{app}\; (\!| u |\!)\; .\mathsf{sn} &&= \mathsf{sn\text{-}Stu}\; ((\!| t |\!)\; .\mathsf{sn})\; ((\!| u |\!)\; .\mathsf{sn})\\ &(\!|\; \mathsf{S}\; \{b = b\}\; |\!)\; .\mathsf{app}\; (\!| t |\!)\; .\mathsf{app}\; (\!| u |\!)\; .\mathsf{app}\; (\!| v |\!) &&= (\!|\mathsf{S}|\!)\; \{b = b\}\; (\!| t |\!)\; (\mathsf{sem\text{-}sn}\; (\!| t |\!))\\ &&&\qquad\quad (\!| u |\!)\; (\mathsf{sem\text{-}sn}\; (\!| u |\!))\\ &&&\qquad\quad (\!| v |\!)\; (\mathsf{sem\text{-}sn}\; (\!| v |\!))\\ &(\!|\; \mathsf{K}\; |\!)\; .\mathsf{sn} &&= \mathsf{sn\text{-}K}\end{aligned}$$

$$
\begin{array}{ll}
⦅\ \mathsf{K}\ ⦆\ .\mathsf{app}\ ⦅t⦆\ .\mathsf{sn} & = \mathsf{sn{-}Kt}\ (\mathsf{sem{-}sn}\ ⦅t⦆) \\
⦅\ \mathsf{K}\ ⦆\ .\mathsf{app}\ ⦅t⦆\ .\mathsf{app}\ ⦅u⦆ & = ⦅\mathsf{K}⦆\ ⦅t⦆\ (\mathsf{sem{-}sn}\ ⦅t⦆)\ (\mathsf{sem{-}sn}\ ⦅u⦆) \\
⦅\ t \bullet u\ ⦆ & = ⦅\ t\ ⦆\ .\mathsf{app}\ ⦅\ u\ ⦆
\end{array}
$$

Strong normalization is now a simple corollary.

$$
\begin{array}{l}
\mathsf{thm} : (t : \mathsf{Tm}\ a) \to \mathsf{SN}\ t \\
\mathsf{thm}\ t = \mathsf{sem{-}sn}\ ⦅\ t\ ⦆
\end{array}
$$

Q.E.D.

Acknowledgments. This document has been generated from an Agda file using the `agda2lagda` translator and the `agda --latex` backend.