

There is no finite model of the SK combinator calculus

Zhenyun Yin

Supervisor : Ambrus Kaposi

Eötvös Loránd University

May 29, 2024



Overview

- **Goal:** To prove there is no finite non-trivial model in SK combinator calculus using machine verification.
- **Proof assistant:** Agda

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

SK combinator calculus

- **In CL:**
 - $t ::= K \mid S \mid (t\ t)$
 - $(Ku)f = u$
 - $((Sf)g)u = (fu)(gu)$
- Turing-complete language without variables.

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

Combinator calculus syntax

Introduction

Overview

SK combinator calculus

Models

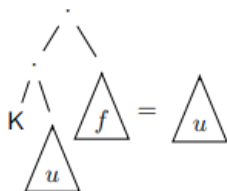
Trivial model

No two-element model

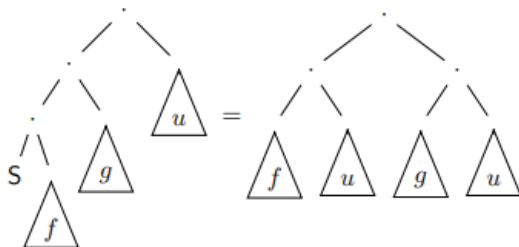
No finite model

Conclusion

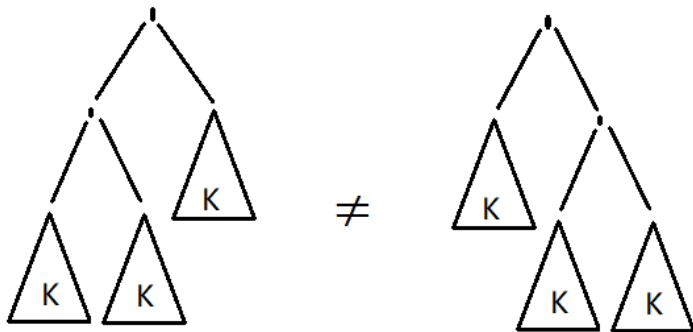
$$Kuf = u$$



$$Sfgu = fu(gu)$$



$$(KK)K \neq K(KK)$$



Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

Programming with combinators

Identity Function (I)

- $I = SKK$
- $Ix = SKKx$
 $= Kx(Kx)$
 $= x$

Function Composition (B)

- $B = S(KS)K$
- $Bfgu = S(KS)Kfgu$
 $= KSf(Kf)gu$
 $= S(Kf)gu$
 $= Kfu(gu)$
 $= f(gu)$

Connection to lambda calculus

- ➊ Introduced by Schönfinkel 1924
- ➋ Lambda calculus: Church 1928
- ➌ They are equivalent (Rosser 1935)
- ➍ Turing machine : Turing 1936
- ➎ Combinatory Logic: Pure, Applied and Typed: Bimbó 2012
- ➏ Combinatory logic and lambda calculus are equal, algebraically (Kaposi, Altenkirch, Sinkarovs and Véghe, FSCD 2023)

➐ In combinator calculus

- $Kuf = u$
- $Sfgu = fu(gu)$

In lambda calculus

- $\lambda uf.u$
- $\lambda fgu.fu(gu)$

A model of combinator calculus in Agda

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

```
record Model : Set1 where
  infixl 5 _._
  field
    Tm : Set
    _._ : Tm → Tm → Tm
    K S : Tm
    Kβ : ∀{u f} → K . u . f ≡ u
    Sβ : ∀{f g u} → S . f . g . u ≡ f . u . (g . u)
```


Trivial model

```
trivial : Model
trivial = record {
  Tm = T ;
  _·_ = λ _ _ → tt ;
  K = tt ;
  S = tt ;
  Kβ = refl ;
  Sβ = refl
}
```

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

Methodology

- **Church Encoding:** Encodes data types as functions in terms of combinators.
- **Proof Techniques:** Utilizes pigeonhole principle.

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

No two-element model

$$\text{proj}_i \cdot u_1 \cdot u_2 \cdot u_3 = u_i$$

- $\text{proj}_1 = B \cdot K \cdot (B \cdot K \cdot I)$
- $\text{proj}_2 = K \cdot (B \cdot K \cdot I)$
- $\text{proj}_3 = K \cdot (K \cdot I)$

$$B \cdot K \cdot a \cdot b \cdot c = a \cdot b$$

$$K \cdot a \cdot b = a$$

$$I \cdot a = a$$

Introduction

[Overview](#)[SK combinator calculus](#)

Models

[Trivial model](#)[No two-element model](#)[No finite model](#)

Conclusion

No two-element model

$$\text{proj}_i \cdot u_1 \cdot u_2 \cdot u_3 = u_i$$

Using the pigeonhole principle:

- Case 1: $\text{proj}_1 = \text{proj}_2$

$$\text{true} = \text{proj}_1 \cdot \text{true} \cdot \text{false} \cdot \text{false} = \text{proj}_2 \cdot \text{true} \cdot \text{false} \cdot \text{false} = \text{false}$$

- Case 2: $\text{proj}_1 = \text{proj}_3$

$$\text{true} = \text{proj}_1 \cdot \text{true} \cdot \text{false} \cdot \text{false} = \text{proj}_3 \cdot \text{true} \cdot \text{false} \cdot \text{false} = \text{false}$$

- Case 3: $\text{proj}_2 = \text{proj}_3$

$$\text{true} = \text{proj}_2 \cdot \text{false} \cdot \text{true} \cdot \text{false} = \text{proj}_3 \cdot \text{false} \cdot \text{true} \cdot \text{false} = \text{false}$$

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

Cases of more elements

$$\text{proj}_i \cdot u_1 \cdot u_2 \dots \cdot u_{n+1} = u_i$$

Ind \ NOP	1	2	3	4
1	I			
2	BKI	KI		
3	BK(BKI)	K(BKI)	K(KI)	
4	BK(BK(BKI))	K(BK(BKI))	K(K(BKI))	K(K(KI))

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion

Proof

All Done!

```

src > SK > Church_encoding.agda
535 module notFiniteModel2
541 where
542
543 almost0 : (j : Fin (suc n)) → Fin (suc n) → Tm
544 almost0 (j) = with x ≤ j
545 ... | inr b → fzero
546 ... | inl a → fsuc fzero
547
548 flatten : ∀ {A : Set B} (n : B) → (Fin n → A) → Vec A n
549 flatten (n = zero) f = []
550 flatten (n = suc n) f = f fzero :: flatten (n - n) (f ∘ fsuc)
551
552 us : (j : Fin (suc n)) → Vec Tm (suc n)
553 us (j) = flatten (almost0 (j))
554
555 projB : (n : B) (x : Fin (suc n)) (us : Vec Tm (suc n)) → proj n x ∙ s us = lookup us x
556 projB zero fzero (u : []) = sB
557 projB (suc n) fzero (u = u' ∙ us) = (B × K ∙ (proj n fzero) ∙ u ∙ u' ∙ s us)
558                                     = (cong (λ x → (x ∙ u') ∙ s us) sB)
559                                     = (((K ∙ (proj n fzero ∙ u) ∙ u') ∙ s us)
560                                     = (cong (λ x → x) sB)
561                                     = ((proj n fzero ∙ u) ∙ s us)
562                                     = (proj n fzero (u ∙ us))
563                                     = refl
564
565 projB (suc n) (fsuc x) (u = u' ∙ us) = (K ∙ (proj n x) ∙ u ∙ u' ∙ s us)
566                                     = (cong (λ x → x ∙ u' ∙ s us) sB)
567                                     = (proj n x ∙ u' ∙ s us)
568                                     = (projB n x (u' ∙ us))
569                                     = refl
570
571 flattenval : ∀ {A : Set B} (n : B) (f : Fin n → A) (x : Fin n) → f x = lookup (flatten f) x
572 flattenval (n = suc n) f fzero = refl
573 flattenval (n = suc n) f (fsuc x) = flattenval (n = n) (f ∘ fsuc) x
574
575 almost0i : ∀ {i j : Fin (suc n)} → i ≤ j → almost0 (j) i = fzero
576 almost0i (i)(j) i ≤ j with i ≤ j
577 ... | inr b → refl
578 ... | inl a with i ≤ j
579 ... | ()
580
581 almost0j : ∀ {j : Fin (suc n)} → almost0 (j) j = (fsuc fzero)
582 almost0j (j) with j ≤ j
583 ...
584
585 Agda
586
587 "All Done!"

```

```

src > SK > Church_encoding.agda
535 module notFiniteModel2
541 where
542
543 ... | inr ~j with ~j refl
544 ... | ()
545
546 f : Fin (suc n) → Fin n
547 f x = proj n x
548
549 fromPigeon : B₂ λ i j → i ≤ j → f i = f j
550 fromPigeon = pigeonhole mSucc f
551
552 contra : fzero = fsuc fzero
553 contra with fromPigeon
554 ... | i , j , i ≤ j , fi=fj =
555   fzero
556   = (sym (almost0i i j))
557   = (flattenval almost0 i)
558   lookup us i
559   = (sym (projB (suc (suc n)) i (us (j))))
560   ((proj (suc (suc n)) i) ∙ s us)
561   = (cong (λ x → x ∙ s us) fi=fj)
562   ((proj (suc (suc n)) j) ∙ s us)
563   = (projB (suc (suc n)) j (us (j)))
564   lookup us j
565   = (sym (flattenval almost0 j))
566   almost0 j
567   = (almost0j (j) j)
568   refl
569
570 bot : ⊥
571 bot with contra
572 bot | ()
573
574 notbool : (m : Model) → let module m = Model in m.Tm = (Fin 2) → ⊥
575 notbool = refl - notboolModel2.bot _ K S sB sB
576 where
577   open Model m
578
579 notfinite : (m : Model) (n : B) → let module m = Model in m.Tm = (Fin (suc (suc n))) → ⊥
580 notfinite record { Tm = (Fin (suc (suc n))) ; u' = 's ; K = K₁ ; S = S₁ ; sB = sB₁ ; sB = sB₁ }
581 notFiniteModel2.bot m _ s₁ K₁ S₁ sB₁ sB₁
582
583
584
585

```

Conclusion

- **Main Result:** There is no finite non-trivial model of the SK combinator calculus.
- **Impact:** Enhances understanding of combinatory logic with machine-verifiable proofs.
- **Next Steps:** Explore infinite models.

Introduction

Overview

SK combinator calculus

Models

Trivial model

No two-element model

No finite model

Conclusion