

Combinator Weak Normalization by Tait Computability

Randy Pollack

Version of December 2, 2011

Outline

- 1 Hilbert Style Logic and the Deduction Theorem
- 2 Tait Computability Proves Normalization

Outline

- 1 Hilbert Style Logic and the Deduction Theorem
- 2 Tait Computability Proves Normalization

Hilbert style (combinator) Minimal Implicational Logic

- Recall the **natural deduction** rules for STLC:

$$\frac{\Gamma \text{ valid} \quad p:A \in \Gamma}{\Gamma \vdash p : A} \quad (\text{ELIM}) \frac{\Gamma \vdash b : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash ba : B}$$

$$(\text{INTRO}) \frac{\Gamma, p:A \vdash b : B}{\Gamma \vdash [p]b : A \rightarrow B}$$

- Computation on terms by β -reduction.
- Hilbert style** combinator presentation of the same logic:

$$\frac{\Gamma \text{ valid} \quad p:A \in \Gamma}{\Gamma \vdash_H p : A} \quad (\text{MP}) \frac{\Gamma \vdash_H b : A \rightarrow B \quad \Gamma \vdash_H a : A}{\Gamma \vdash_H ba : B}$$

$$(K) \quad \Gamma \vdash_H k : A \rightarrow B \rightarrow A$$

$$(S) \quad \Gamma \vdash_H s : (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)$$

- Computation on terms?

SK language

- The language of types (propositions) of the H system is the same as the ND system:

$$A ::= P \mid A \rightarrow B$$

where P, Q, \dots are *propositional variables*.

- The term language of the SK system (over a set of term variables p, q, \dots):

$$M ::= p \mid MN \mid k \mid s$$

k, s are constants.

- **No variable binding.** Easy to reason about.

SK computation: *weak reduction*

contractions: $kab > a$ $sabc > ac(bc)$

congruences: $\frac{a > a'}{ab > a'b}$ $\frac{b > b'}{ab > ab'}$

- Terms of the form kab and $sabc$ are called *redexes*.
- k **throws away** an argument; s **duplicates** an argument.
- For any proposition A , there is an MIL proof of $\vdash_H skk : A \rightarrow A$.
 - $I := skk$ is the identity function
 - Let a be any term, and compute:

$$skka > ka(ka) > a.$$

- Subject reduction holds for \vdash_H with $>$.

> is Turing complete

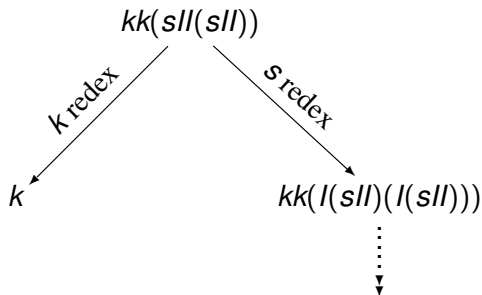
- > is not terminating: let $I \equiv skk$, have

$$sll(sll) > I(sll)(I(sll)) > sll(I(sll)) > sll(sll) > \dots$$

- A fixpoint operator is expressible in the SK calculus.
 - Thus all computable functions are representable.
- But we will show that if $\Gamma \vdash_H a : A$ is provable, then a is terminating.
- There are other computationally complete combinator sets with better properties w.r.t. program size.

Confluence of $>$

- A term can have many reduction sequences.



- Some reduction sequences may terminate while others do not
- $>$ has Church–Rosser (fairly easy).
- Thus normal forms (when they exist) are unique.

ND system can mimic the Hilbert system

- ND already has assumption and MP.
 - All we need is to simulate k, s in the ND system.
- Define k, s as lambda terms

$$k = [p][q]p$$
$$s = [p][q][r]pr(qr)$$

- Easily verify these have the correct reduction and types.

Hilbert system can mimic ND: Deduction Theorem

- \vdash_H already has assumption and MP; must simulate INTRO.
- **Deduction Theorem** There is a function $[_]^*_$ (*combinatory abstraction*) such that

$$\frac{\Gamma, p:A \vdash_H b : B}{\Gamma \vdash_H [p]^*b : A \rightarrow B}$$

is admissible in \vdash_H .

- **Proof** Take $[_]^*_$ to be:

$$\begin{aligned} [p]^*p &= skk \\ [p]^*\alpha &= k\alpha && \alpha = k, s, q, \alpha \neq p \\ [p]^*bc &= s([p]^*b)([p]^*c) \end{aligned} \quad \square$$

- Easily verify that $([p]^*b)c >^*_w [c/p]b$.
- Other definitions of combinatory abstraction also work; some have better properties.

Outline

- 1 Hilbert Style Logic and the Deduction Theorem
- 2 Tait Computability Proves Normalization**

Types and terms

- Simple types over countably many atomic propositions.

```

Inductive prop : Set :=
  | atom: nat -> prop      (* countably many atomic
  | arrow: prop -> prop -> prop .
Notation "p ~> q" := (arrow p q) (... right ...).

```

- Terms: k, s and apply, plus **typed variables**.

```

Inductive term : Set :=
  | k: term
  | s: term
  | v: nat -> prop -> term
  | app: term -> term -> term.
Notation "a & b" := (app a b) (... left ...).

```

- Think of expression $(v\ n\ p)$ as variable v_n^p .

Untyped reduction rules

- *sk* rules as before:

contractions: $kab > a$ $sabc > ac(bc)$

congruences: $\frac{a > a'}{ab > a'b}$ $\frac{b > b'}{ab > ab'}$

```

Inductive red : term -> term -> Prop :=
| kred: forall a b, red (k & a & b) a
| sred: forall a b c,
    red (s & a & b & c) ((a & c) & (b & c))
| app_lcong: forall a a' b, (red a a') ->
    red (a & b) (a' & b)
| app_rcong: forall a a' b, (red a a') ->
    red (b & a) (b & a').

```

Notation "a --> b" := (red a b) (at level 79).

Definition of **Weakly Normalizes**

$WNorm(a)$ holds if a has **some** reduction path to a normal form.

- Rules for constants

$$\frac{}{WNorm(k)} \quad \frac{}{WNorm(s)}$$

- Rules for congruence

$$\frac{WNorm(a)}{WNorm(ka)} \quad \frac{WNorm(a)}{WNorm(sa)} \quad \frac{WNorm(a) \quad WNorm(b)}{WNorm(sab)}$$

- The step case: if a terminating reduction sequence is extended (backwards) by one step, it still terminates.

$$\frac{WNorm(b) \quad a > b}{WNorm(a)}$$

- Not done yet: neutral terms. if v is a variable and M_1, \dots, M_n are normalizing, then $vM_1 \dots M_n$ is normalizing.

Definition of Weakly Normalizes: Neutral terms

- *Neutral terms* are those that cannot interact with any evaluation context.
- The normalizing neutral terms are mutually defined with the normalizing terms.

$$\frac{}{WNorm(k)} \quad \frac{}{WNorm(s)} \quad \frac{Neut(a)}{WNorm(a)}$$

$$\frac{WNorm(a)}{WNorm(ka)} \quad \frac{WNorm(a)}{WNorm(sa)} \quad \frac{WNorm(a) \quad WNorm(b)}{WNorm(sab)}$$

$$\frac{WNorm(b) \quad a > b}{WNorm(a)}$$

$$\frac{}{Neut(v_n^p)} \quad \frac{Neut(N) \quad WNorm(M)}{Neut(NM)}$$

Typing rules; as expected

$$\frac{}{\vdash_H v_n^A : A} \quad (\text{MP}) \frac{\vdash_H b : A \rightarrow B \quad \vdash_H a : A}{\vdash_H ba : B}$$

$$(K) \quad \vdash_H k : A \rightarrow B \rightarrow A$$

$$(S) \quad \vdash_H s : (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)$$

```

Inductive thm : term -> prop -> Prop :=
| K: forall p q, thm k (p ~> q ~> p)
| S: forall p q r,
    thm s ((p ~> q ~> r) ~> (p ~> q) ~> (p ~> r))
| V: forall n p, thm (v n p) p
| MP: forall a b p q (lp: thm a (p ~> q))
      (rp: thm b p),
      (*****)
      (thm (a & b) q).
  
```

The main theorem

- Now we can state:

Theorem AllWNorm:

$$\text{forall } p \ M, \ (\text{thm } M \ p) \ \rightarrow \ \text{WNorm } M.$$

- To believe that simply typed terms of the SK calculus are normalizing, you must:
 - understand the definitions above and
 - believe that the definitions and formal theorem mean what I claim.
- What follows is a proof of the theorem checked in Coq:
 - If you trust Coq, you can believe the theorem without understanding what follows.

The key definition: Tait Computability

```

Fixpoint Comp p M {struct p} : Prop :=
  match p with
  | atom n => thm M (atom n) /\ WNorm M
  | q ~> r => thm M (q ~> r) /\ WNorm M
      /\ (forall N, Comp q N -> Comp r (M & N))
  end.

```

- Definition by structural recursion on **type**
 - q and r are structural components of $q \rightarrow r$.
- Normalizing terms of atomic type are computable.
- A term M is computable at type $p \rightarrow q$ if
 - M has type $p \rightarrow q$
 - $WNorm(M)$
 - for all N computable at type p , MN is computable at type q .

This is a subtle definition

Can we define the *graph* of the computability function as an inductive relation:

```

Inductive COMP : prop -> term -> Prop :=
  | cAtm : forall M n,
      thm M (atom n) -> WNorm M -> COMP (atom n) M
  | cArr : forall q r M,
      thm M (q ~> r) ->
      WNorm M ->
      (forall N, {COMP q N} -> COMP r (M & N)) ->
      (* ***** *)
      COMP (q ~> r) M.
  
```

This definition is not accepted because of the negative occurrence of `COMP q N`.

Some simple properties of Computability

These are one-line proofs in Coq: mostly by definition.

```
Lemma CompThm: forall p M, (Comp p M) -> thm M p.
```

```
Lemma CompWNorm: forall p M, (Comp p M) -> WNorm M.
```

```
Lemma appPreserveComp: forall p q M N,
  (Comp (p ~> q) M) -> (Comp p N) -> Comp q (M & N).
```

A key property; simple proof by induction on p :

```
Lemma ExpandPreserveComp:
  forall p M N, thm M p -> (M --> N) ->
    Comp p N -> Comp p M.
```

Basic term constructors are computable

k and s are computable: expand definitions and use the previous lemmas.

Lemma kComp: forall p q, Comp (p \sim > q \sim > p) k.

Lemma sComp: forall p q r,
 Comp ((p \sim > q \sim > r) \sim > (p \sim > q) \sim > (p \sim > r)) s.

All neutral terms are computable: proof by induction on p .

Lemma NeutComp: forall p N, thm N p \rightarrow
 Neutral N \rightarrow Comp p N.

This stumped me for a while, but it is an easy proof once you state the right property (thanks Conor McBride).

Putting it all together: the normalization proof

Lemma AllComp: forall M p, (thm M p) -> Comp p M.

Proved by induction on the premise. There are 4 cases:

- 1 k : use Lemma kComp.
- 2 s : use Lemma sComp.
- 3 variable: use Lemma NeutComp
- 4 application: use Lemma appPreserveComp.

The Main Theorem follows trivially from this.

Theorem AllWNorm: forall p M, (thm M p) -> WNorm M.

Thus simply typed sk terms are normalizing
equivalently, simply typed lambda terms are normalizing.

More things to do

- We have defined *Normalizing*
 - Could also define *Normal form* . . .
 - prove that normalizing terms actually reduce to a normal form (using reflexive-transitive-closure of \rightarrow).
- Actually do normalization inside Coq.
- Extract a normalizer program in OCaml or Haskell.
- Prove subject reduction for \vdash_H .
- Prove the deduction theorem.