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
Recall the natural deduction rules for STLC:

\[
\frac{\Gamma \text{ valid} \quad p : A \in \Gamma}{\Gamma \vdash p : A} \quad \frac{\Gamma \vdash b : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash b \ a : B}
\]  

\[
\frac{\Gamma, p : A \vdash b : B}{\Gamma \vdash \lambda p \ b : A \rightarrow B}
\]

- Computation on terms by \( \beta \)-reduction.

**Hilbert style** combinator presentation of the same logic:

\[
\frac{\Gamma \text{ valid} \quad p : A \in \Gamma}{\Gamma \vdash_{\text{H}} p : A} \quad \frac{\Gamma \vdash_{\text{H}} b : A \rightarrow B \quad \Gamma \vdash_{\text{H}} a : A}{\Gamma \vdash_{\text{H}} b \ a : B}
\]

\[
\frac{}{\Gamma \vdash_{\text{H}} k : A \rightarrow B \rightarrow A}
\]

\[
\frac{}{\Gamma \vdash_{\text{H}} s : (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C)}
\]

- 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 \( P, Q, \ldots \) are propositional variables.

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

  \[ M ::= p \mid M N \mid k \mid s \]

  \( k, s \) are constants.

- No variable binding. Easy to reason about.
SK computation: weak reduction

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

congruences: \( a > a' \) \( b > b' \) \( ab > a'b \) \( ab > ab' \)

- 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 $l \equiv skk$, have

$$sll(sll) > l(sll)(l(sll)) > sl(l(sll)) > sll(sll) > \ldots$$

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 $\rightarrow$

- A term can have many reduction sequences.

\[ kk(sI I(sI I)) \]

\[ \text{K redex} \quad \text{s redex} \]

\[ k \quad kk(I(sI I)(I(sI I))) \]

- Some reduction sequences may terminate while others do not.
- $\rightarrow$ 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

\[
\begin{align*}
\Gamma, p: A & \vdash_H b : B \\
\Gamma & \vdash_H [p]*b : A \rightarrow B
\end{align*}
\]

is admissible in \( \vdash_H \).
- **Proof** Take \( [\_]*\_ \) to be:

\[
\begin{align*}
[p]*p &= skk \\
[p]*\alpha &= k\alpha \\
[p]*bc &= s([p]*b)([p]*c)
\end{align*}
\]

\( \alpha = k, s, q, \alpha \neq p \)

Easily verify that \( ( [p]*b )c \ast [c/p]b \).
- 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:**
  \[ kab > a \quad sabc > ac(bc) \]

  **congruences:**
  \[
  \frac{a > a'}{ab > a'b} \quad \frac{b > b'}{ab > ab'}
  \]

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**

\[ \text{WNorm}(a) \] holds if \( a \) has *some* reduction path to a normal form.

- **Rules for constants**
  
  \[
  \begin{array}{c}
  \text{WNorm}(k) \\
  \text{WNorm}(s)
  \end{array}
  \]

- **Rules for congruence**
  
  \[
  \begin{array}{c}
  \text{WNorm}(a) \\
  \text{WNorm}(k a) \\
  \text{WNorm}(s a) \\
  \text{WNorm}(s a b)
  \end{array}
  \]

- **The step case:** if a terminating reduction sequence is extended (backwards) by one step, it still terminates.
  
  \[
  \begin{array}{c}
  \text{WNorm}(b) \\
  a > b
  \end{array}
  \]

  \[
  \text{WNorm}(a)
  \]

- **Not done yet: neutral terms.** If \( v \) is a variable and \( M_1, \ldots, M_n \) are normalizing, then \( vM_1 \ldots M_n \) 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.

\[
\begin{align*}
\text{WNorm}(k) & \quad \text{WNorm}(s) & \quad \text{WNorm}(a) \\
\text{WNorm}(a) & \quad \text{WNorm}(s) & \quad \text{WNorm}(k a) \\
\text{WNorm}(a) & \quad \text{WNorm}(s a) & \quad \text{WNorm}(s a b) \\
\text{WNorm}(b) & \quad a > b & \quad \text{WNorm}(a) \\
\text{Neut}(v_n^p) & \quad \text{Neut}(N) & \quad \text{WNorm}(M) \\
\text{Neut}(NM) & \quad \text{Neut}(NM) &
\end{align*}
\]
Typing rules; as expected

\[ \frac{\vdash \nu_n^A : A}{\vdash H} \quad \text{(MP)} \quad \frac{\vdash b : A \rightarrow B \quad \vdash a : A}{\vdash a \cdot b : B} \]

(K) \[ \vdash_H k : A \rightarrow B \rightarrow A \]

(S) \[ \vdash_H s : (A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow (A \rightarrow C) \]

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).
Now we can state:

Theorem AllWNorm:
\[ \forall p \ M, (\text{thm} \ M \ p) \rightarrow \text{WNorm} \ M. \]

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

\[ \text{Fixpoint } \text{Comp} \ p \ M \ \{\text{struct } p\} : \text{Prop} := \]
\[ \text{match } p \text{ with} \]
\[ | \text{atom } n \Rightarrow \text{thm} \ M \ (\text{atom } n) \ \land \ W\text{Norm} \ M \]
\[ | q \rightarrow r \Rightarrow \text{thm} \ M \ (q \rightarrow r) \ \land \ W\text{Norm} \ M \]
\[ \land \ (\forall N, \text{Comp} \ q \ N \rightarrow \text{Comp} \ r \ (M \land N)) \]
\end{match}

- 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 \( p \rightarrow q \)
  - \( W\text{Norm}(M) \)
  - 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) -> WN Norm M -> COMP (atom n) M
  | cArr : forall q r M, 
    thm M (q ~> r) -> WN Norm 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 \text{kComp}: \forall p q, \text{Comp} (p \to q \to p) k.

Lemma \text{sComp}: \forall p q r, 
\text{Comp} ((p \to q \to r) \to (p \to q) \to (p \to r)) s.

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

Lemma \text{NeutComp}: \forall p N, \text{thm} N p \to 
\text{Neutral} N \to \text{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: \( \forall M \ p, \ (\text{thm} \ M \ p) \rightarrow \text{Comp} \ p \ M. \)

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

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 Haskel.
- Prove subject reduction for \(\vdash_H\).
- Prove the deduction theorem.