Combinator Weak Normalization by Tait Computability

Randy Pollack

Version of December 2, 2011

Outline

Outline

Hilbert style (combinator) Minimal Implicational Logic

- Recall the natural deduction rules for STLC:

$$\frac{\Gamma \text{ valid} \qquad p{:}A \in \Gamma}{\Gamma \vdash p : A} \qquad (\text{ELIM})\ \frac{\Gamma \vdash b : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash b\,a : B}$$

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

  - Computation on terms by β-reduction.

- Hilbert style combinator presentation of the same logic:

$$\frac{\Gamma \text{ valid} \qquad p{:}A \in \Gamma}{\Gamma \vdash_{\mathsf{H}} p : A} \qquad (\text{MP})\ \frac{\Gamma \vdash_{\mathsf{H}} b : A \to B \qquad \Gamma \vdash_{\mathsf{H}} a : A}{\Gamma \vdash_{\mathsf{H}} b\,a : B}$$

$$(\text{K}) \qquad \Gamma \vdash_{\mathsf{H}} k : A \to B \to A$$

$$(\text{S}) \qquad \Gamma \vdash_{\mathsf{H}} s : (A \to B \to C) \to (A \to B) \to (A \to C)$$

  - 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: $kab > a \qquad sabc > ac(bc)$

congruences: $\dfrac{a > a'}{ab > a'b} \qquad \dfrac{b > b'}{ab > ab'}$

- 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 $\vdash_{\mathrm{H}} skk : A \to A$.
  - $I := skk$ is the identity function
  - Let a be any term, and compute:

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

- Subject reduction holds for $\vdash_{\mathrm{H}}$ with $>$.

$>$ is Turing complete

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

$$sII(sII) > I(sII)(I(sII)) > sII(I(sII)) > sII(sII) > \dots$$

- A fixpoint operator is expressible in the SK calculus.
  - Thus all computable functions are representable.
- But we will show that if $\Gamma \vdash_{\mathrm{H}} a : A$ 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.

$$
\begin{tikzcd}
 & kk(sII(sII)) \arrow[dl, "k\ \text{redex}"'] \arrow[dr, "s\ \text{redex}"] & \\
k & & kk(I(sII)(I(sII))) \arrow[d, dotted] \\
 & & \,
\end{tikzcd}
$$

- 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

$$\begin{aligned} k &= [p][q]p \\ s &= [p][q][r]pr(qr) \end{aligned}$$

- 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:

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

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

Outline

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: $kab > a \qquad sabc > ac(bc)$

congruences: $\dfrac{a > a'}{ab > a'b} \qquad \dfrac{b > b'}{ab > ab'}$

```
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

$\mathit{WNorm}(a)$ holds if a has some reduction path to a normal form.

- Rules for constants

$$\frac{}{\mathit{WNorm}(k)} \qquad \frac{}{\mathit{WNorm}(s)}$$

- Rules for congruence

$$\frac{\mathit{WNorm}(a)}{\mathit{WNorm}(k\ a)} \qquad \frac{\mathit{WNorm}(a)}{\mathit{WNorm}(s\ a)} \qquad \frac{\mathit{WNorm}(a) \qquad \mathit{WNorm}(b)}{\mathit{WNorm}(s\ a\ b)}$$

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

$$\frac{\mathit{WNorm}(b) \qquad a > b}{\mathit{WNorm}(a)}$$

- 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.

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

$$\frac{WNorm\,(a)}{WNorm\,(k\;a)} \qquad \frac{WNorm\,(a)}{WNorm\,(s\;a)} \qquad \frac{WNorm\,(a) \qquad WNorm\,(b)}{WNorm\,(s\;a\;b)}$$

$$\frac{WNorm\,(b) \qquad a > b}{WNorm\,(a)}$$

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

Typing rules; as expected

$$\frac{}{\vdash_{\mathrm{H}} v_n^A : A} \qquad (\mathrm{MP})\ \frac{\vdash_{\mathrm{H}} b : A \to B \qquad \vdash_{\mathrm{H}} a : A}{\vdash_{\mathrm{H}} b\,a : B}$$

(K) $\vdash_{\mathrm{H}} k : A \to B \to A$

(S) $\vdash_{\mathrm{H}} s : (A \to B \to C) \to (A \to B) \to (A \to C)$

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

- To believe that simply typed terms of the SK calculus are normalizing, you must:
  - understand the defintions 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 $q \to r$.
- Normalizing terms of atomic type are computable.
- A term M is computable at type $p \to q$ if
  - M has type $p \to q$
  - $WNorm(M)$
  - 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 ~> q ~> p) k.
```

```
Lemma sComp: forall p q r,
  Comp ((p ~> q ~> r) ~> (p ~> q) ~> (p ~> r)) s.
```

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

```
Lemma NeutComp: forall p N, thm N p ->
   Neutral N -> 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 $-->$).
- Actually do normalization inside Coq.
- Extract a normalizer program in OCaml or Haskel.
- Prove subject reduction for $\vdash_H$.
- Prove the deduction theorem.