Normalization

Advanced Compiler Construction and Program Analysis

Lecture 5

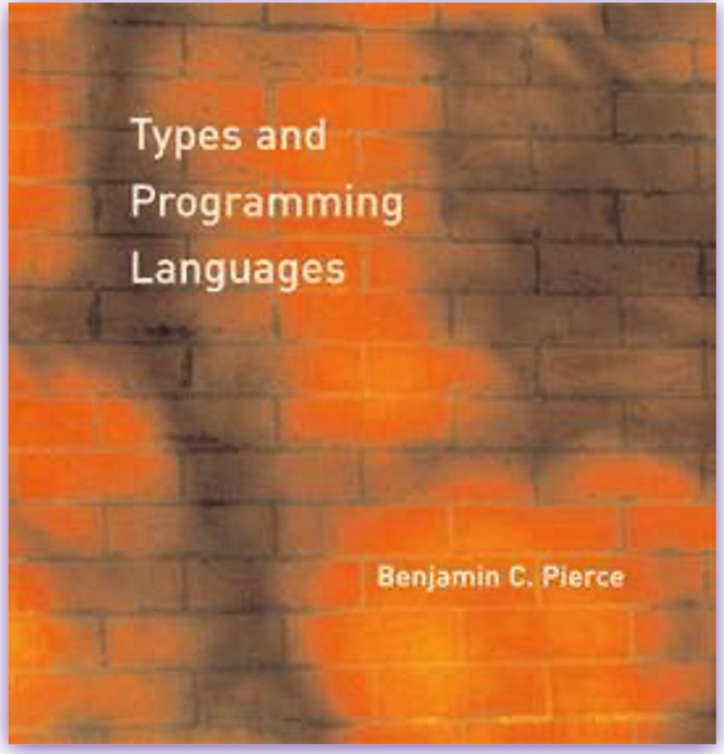
Innopolis University, Spring 2022

The topics of this lecture are covered in detail in...

Benjamin C. Pierce.

Types and Programming Languages

MIT Press 2002



Normalizable terms

If evaluation of a term `t` halts in a finite number of steps, then we say that term `t` is ***normalizable***.

We will see that **every well-typed term is normalizable** in simply typed λ-calculus over a single base type.

Turing-completeness vs Normalization

Unlike type safety, normalization property is not common to programming languages (indeed, it is incompatible with Turing completeness). However, it is still Turing completeness is **undesirable** in many places, e.g.:

1. Configuration languages (e.g. YAML, Ansible)
2. Markup languages (e.g. HTML, CSS, Markdown)
3. Excel formulas
4. Types (in more advanced type systems) and more

Check out some surprisingly Turing-complete systems at https://www.gwern.net/Turing-complete

Simply typed λ-calculus over a single base type

`t ::=`		***terms***
	`x`	*variable*
	`λx:T.t`	*abstraction*
	`t t`	*application*
`T ::=`		***types***
	`A`	*base type A*
	`T → T`	*function type*

$$(\lambda x.t_1)\ t_2 \longrightarrow [x \mapsto t_2]t_1$$

$$\Gamma,\ x{:}T \vdash x : T$$

$$\frac{\Gamma,\ x{:}T_1 \vdash t : T_2}{\Gamma \vdash (\lambda x{:}T_1.\ t) : T_1 \rightarrow T_2}$$

$$\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\ t_2 : T_2}$$

Goal: show that if $\vdash t : T$, then t is normalizable.

Simply typed λ-calculus over a single base type

Exercise 5.1. Show why it is impossible to prove normalization for simply typed λ-calculus using induction on the size of terms.

Simply typed λ-calculus over a single base type

Exercise 5.1. Show why it is impossible to prove normalization for simply typed λ-calculus using induction on the size of terms.

We need a stronger inductive hypothesis!

Reducibility candidates

Definition 5.2. For each type T, we define a set `R[T]` of closed normalizing terms of type T:

Reducibility candidates

Definition 5.2. For each type T, we define a set `R[T]` of closed normalizing terms of type T:

1. `t` ∈ `R[A]` <u>iff</u> `t` has type A and `t` halts;

Reducibility candidates

Definition 5.2. For each type T, we define a set R[T] of closed normalizing terms of type T:

1. $t \in$ R[A] <u>iff</u> t has type A and t halts;
2. $t \in$ R[$T_1 \to T_2$] <u>iff</u>
   - t has type $T_1 \to T_2$,
   - t halts,
   - for any $s \in$ R[T_1], we have (t s) $\in$ R[T_2].

Reducibility candidates halt

Lemma 5.3. If `t` ∈ `R[T]`, then `t` halts.

Proof. Follows directly from the definition of `R[T]`.

Reducibility candidates: preservation

Lemma 5.4. If $\mathtt{t:T}$ and $\mathtt{t} \longrightarrow \mathtt{u}$, then $\mathtt{t} \in \mathtt{R[T]}$ iff $\mathtt{u} \in \mathtt{R[T]}$.

Proof. By induction on the structure of type **T**.
Note that **t** halts iff **u** halts. If **T=A**, then there is nothing else to prove.
If $\mathbf{T=T_1 \to T_2}$, then

1. Suppose $\mathtt{t} \in \mathtt{R[T_1 \to T_2]}$ and $\mathtt{s} \in \mathtt{R[T_1]}$, then $\mathtt{(t\ s)} \in \mathtt{R[T_2]}$.

   But $\mathtt{(t\ s)} \longrightarrow \mathtt{(u\ s)}$, so by induction hypothesis, $\mathtt{(u\ s)} \in \mathtt{R[T_2]}$.
   Thus, we have $\mathtt{u} \in \mathtt{R[T_1 \to T_2]}$.

2. In the “only if” direction the proof is analogous.

Reducibility candidates: open terms

Lemma 5.5. If $x_1 : T_1, \ldots, x_n : T_n \vdash t : T$ and $v_1, \ldots, v_n$ are values of types $T_1, \ldots, T_n$, such that $v_i \in R[T_i]$ for each i, then

$$[x_1 \mapsto v_1] \ldots [x_n \mapsto v_n] t \in R[T].$$

Reducibility candidates: open terms

Lemma 5.5. If $x_1 : T_1, \ldots, x_n : T_n \vdash t : T$ and $v_1, \ldots, v_n$ are values of types $T_1, \ldots, T_n$, such that $v_i \in R[T_i]$ for each i, then

$$[x_1 \mapsto v_1] \ldots [x_n \mapsto v_n] t \in R[T].$$

Proof. By induction on the derivation of the ***typing relation***.

Reducibility candidates: open terms (case 1)

Lemma 5.5. If $x_1{:}T_1, \ldots, x_n{:}T_n \vdash t{:}T$ and $v_1, \ldots, v_n$ are values of types $T_1, \ldots, T_n$, such that $v_i \in R[T_i]$ for each i, then $[x_1 \mapsto v_1]\ldots[x_n \mapsto v_n]t \in R[T]$.

$$\boxed{\Gamma,\ x{:}T \vdash x : T}$$

Case 1 (variables)

Proof is immediate (*can you articulate exactly how?*).

Reducibility candidates: open terms (case 2)

Lemma 5.5. If $x_1 : T_1, \ldots, x_n : T_n \vdash t : T$ and $v_1, \ldots, v_n$ are values of types $T_1, \ldots, T_n$, such that $v_i \in R[T_i]$ for each i, then $[x_1 \mapsto v_1] \ldots [x_n \mapsto v_n]t \in R[T]$.

Case 2 (abstraction)

$$\frac{\Gamma,\ x : T_1 \vdash t : T_2}{\Gamma \vdash (\lambda x{:}T_1.\ t) : T_1 \to T_2}$$

Complete the proof.

Reducibility candidates: open terms (case 2)

Lemma 5.5. If $x_1{:}T_1,\ldots,x_n{:}T_n \vdash t{:}T$ and $v_1,\ldots,v_n$ are values of types $T_1,\ldots,T_n$, such that $v_i \in R[T_i]$ for each i, then $[x_1 \mapsto v_1]\ldots[x_n \mapsto v_n]t \in R[T]$.

Case 3 (application)

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\ t_2 : T_2}$$

Complete the proof.

Strong normalization of simply typed λ-calculus

Theorem 5.6. If `⊢ t : T`, then `t` is normalizable.

Proof. Direct corollary of Lemma 5.5 and Lemma 5.3.

Strong normalization proof exercise

Exercise 5.7. Extend the proof of Theorem 5.6 to simply typed lambda calculus extended with **booleans** and **tuples**.

Five stages of YAML

1. Configuration languages are too complex;
   YAML is much simpler and easier to understand.
2. Declarative YAML configuration is brilliant.
3. Lots of our things look similar, we have too much copy and pasted YAML.
4. We've written a tool which uses templates
   and parameters to dynamically generate our YAML
5. The declarative YAML format now supports conditional, iteration, and
   inheritance syntax; it is now turing complete.

https://brokenco.de/2018/08/15/five-stages-of-yaml.html

Dhall: a non-repetitive alternative to YAML

Dhall

```
{- More than one user?  Use a function! -}

let makeUser = \(user : Text) ->
      let home       = "/home/${user}"
      let privateKey = "${home}/.ssh/id_ed25519"
      let publicKey  = "${privateKey}.pub"
      in  { home, privateKey, publicKey }
    {- Add another user to this list -}
in  [ makeUser "bill"
    , makeUser "jane"
    ]
```

YAML

```

- home: /home/bill
  privateKey: /home/bill/.ssh/id_ed25519
  publicKey: /home/bill/.ssh/id_ed25519.pub
- home: /home/jane
  privateKey: /home/jane/.ssh/id_ed25519
  publicKey: /home/jane/.ssh/id_ed25519.pub

```

https://dhall-lang.org/

Summary

- ❑ Normalization vs Turing-completeness
- ❑ Normalization proof for simple types

See you next time!