# 编程语言的设计原理

# Design Principles of Programming Languages

Haiyan Zhao,  Di Wang
趙海燕，王迪

Peking University, Spring Term 2023

# Chapter 3: Untyped Arithmetic Expressions

A small language of Numbers and Booleans

Basic aspects of programming languages

# Introduction

Grammar

Programs

Evaluation

# Grammar (Syntax)

| | |
|---|---|
| t ::= | terms: |
| true | constant true |
| false | constant false |
| if t then t else t | conditional |
| 0 | constant zero |
| succ t | successor |
| pred t | predecessor |
| iszero t | zero test |

t: metavaraible in the right-hand side (non-terminal symbol)

For the moment, the words *term* and *expression* are used interchangeably

# Programs and Evaluations

- A *program* in the language is just *a term* built from *the forms* given by the grammar.

if false then 0 else 1 (1 = succ 0)

→ 1

iszero (pred (succ 0))

→ true

succ(succ(succ(0)))

→?

# Syntax

Many ways of defining syntax (besides grammar)

# Terms, Inductively

The set of terms is the smallest set T such that

1. {true, false, 0} $\subseteq$ T;
2. if $t_1 \in T$,

   then {succ $t_1$, pred $t_1$, iszero $t_1$} $\subseteq$ T;

1. if $t_1 \in T$, $t_2 \in T$, and $t_3 \in T$,

   then if $t_1$ then $t_2$ else $t_3 \in T$.

# Terms, by Inference Rules

The set of terms is defined by the following *rules*:

$$\texttt{true} \in \mathcal{T} \qquad \texttt{false} \in \mathcal{T} \qquad 0 \in \mathcal{T}$$

$$\frac{t_1 \in \mathcal{T}}{\texttt{succ}\ t_1 \in \mathcal{T}} \qquad \frac{t_1 \in \mathcal{T}}{\texttt{pred}\ t_1 \in \mathcal{T}} \qquad \frac{t_1 \in \mathcal{T}}{\texttt{iszero}\ t_1 \in \mathcal{T}}$$

$$\frac{t_1 \in \mathcal{T} \qquad t_2 \in \mathcal{T} \qquad t_3 \in \mathcal{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \in \mathcal{T}}$$

each rule: "If we have established the statements in the premise(s) listed above the line, then we may derive the conclusion below the line

Inference rules = Axioms + Proper rules

# Terms, Concretely

For each natural number i, define a set Si as follows:

$$\begin{aligned} S_0 &= \varnothing \\ S_{i+1} &= \{\texttt{true}, \texttt{false}, 0\} \\ &\cup \{\texttt{succ}\ t_1, \texttt{pred}\ t_1, \texttt{iszero}\ t_1 \mid t_1 \in S_i\} \\ &\cup \{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \mid t_1, t_2, t_3 \in S_i\}. \end{aligned}$$

Finally, let

$$S = \bigcup_i S_i.$$

Exercise [**]: How many elements does $S_3$ have?

Proposition: T = S

# Induction on Terms

Inductive definitions

Inductive proofs

# Inductive Definitions

The set of *constants* appearing in a term *t*, written *Consts(t)*, is defined as:

$$
\begin{aligned}
Consts(\texttt{true}) &= \{\texttt{true}\} \\
Consts(\texttt{false}) &= \{\texttt{false}\} \\
Consts(\texttt{0}) &= \{\texttt{0}\} \\
Consts(\texttt{succ } t_1) &= Consts(t_1) \\
Consts(\texttt{pred } t_1) &= Consts(t_1) \\
Consts(\texttt{iszero } t_1) &= Consts(t_1) \\
Consts(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3) &= Consts(t_1) \cup Consts(t_2) \cup Consts(t_3)
\end{aligned}
$$

# Inductive Definitions

The size of a term *t*, written *size(t)*, is defined as follows:

$$
\begin{aligned}
size(\texttt{true}) &= 1 \\
size(\texttt{false}) &= 1 \\
size(\texttt{0}) &= 1 \\
size(\texttt{succ } t_1) &= size(t_1) + 1 \\
size(\texttt{pred } t_1) &= size(t_1) + 1 \\
size(\texttt{iszero } t_1) &= size(t_1) + 1 \\
size(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3) &= size(t_1) + size(t_2) + size(t_3) + 1
\end{aligned}
$$

# Inductive Definitions

The *depth* of a term *t*, written *depth(t)*, is defined as follows:

$$
\begin{aligned}
depth(\mathtt{true}) &= 1\\
depth(\mathtt{false}) &= 1\\
depth(\mathtt{0}) &= 1\\
depth(\mathtt{succ}\ t_1) &= depth(t_1) + 1\\
depth(\mathtt{pred}\ t_1) &= depth(t_1) + 1\\
depth(\mathtt{iszero}\ t_1) &= depth(t_1) + 1\\
depth(\mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3) &= \max(depth(t_1), depth(t_2), depth(t_3)) + 1
\end{aligned}
$$

# Inductive Proof

**Lemma.**

The number of distinct *constants* in a term *t* is no greater than the *size* of *t*:

$| \text{Consts}(t) | \leq \text{size}(t)$

**Proof.** By induction over the *depth* of *t.*

- Case *t* is a constant : $|\text{Consts}(t)| = |\{t\}| = 1 = \text{size}(t)$.
- Case *t* is pred t1, succ t1, or iszero t1

  By the induction hypothesis, $|\text{Consts}(t1)| \leq \text{size}(t1)$, and we have : $|\text{Consts}(t)| = |\text{Consts}(t1)| \leq \text{size}(t1) < \text{size}(t)$.
- Case *t* is if t1 then t2 else t3

  ?

# Inductive Proof

**Theorem** [Structural Induction]

If, for each term *s*,

given P(r) for all immediate subterms *r* of *s*

we can show P(s),

then P(s) holds for all *s*.

suppose P is a predicate on terms.

# Semantic Styles

Three basic approaches

# Operational Semantics

- Operational semantics specifies the *behavior* of a programming language by defining a simple abstract machine for it.

- An example (often used in this course):
  - terms as *states*
  - *transition from one state to another* as *simplification* (behavior*)*
  - meaning of *t* is *the final state* starting from the state corresponding to *t*

# Denotational Semantics

- Giving denotational semantics for a language consists of
  - finding a *collection of semantic domains*, and then
  - defining an *interpretation function* mapping *terms* into *elements of these domains*.

- Main advantage: It abstracts from the gritty details of evaluation and highlights *the essential concepts* of the language.

# Axiomatic Semantics

- Axiomatic methods take the *laws* (properties) themselves *as the definition of the language.*

- The meaning of a *term* is just *what* can be proved about it.

  – They focus attention on *the process of reasoning* about programs.

  – Hoare logic: define the meaning of imperative languages

# Evaluation

Evaluation relation (small-step/big-step)

Normal form

Confluence and termination

# Evaluation on Booleans

*Syntax*

| | | | |
|---|---|---|---|
| `t` | ::= | | *terms:* |
| | | `true` | *constant true* |
| | | `false` | *constant false* |
| | | `if t then t else t` | *conditional* |
| `v` | ::= | | *values:* |
| | | `true` | *true value* |
| | | `false` | *false value* |

*Evaluation* $\boxed{t \longrightarrow t'}$

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \quad \text{(E-IfTrue)}$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \quad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \quad \text{(E-If)}$$

"t evaluates to t' in one step

# One-step Evaluation Relation

- The *one-step evaluation relation* → is the *smallest binary relation* on terms satisfying the three rules in the previous slide.

- When the *pair (t, t')* is in the evaluation relation, we say that

  "t → t' is *derivable*."

# Derivation Tree

- "if t then false else false → if u then false else false" is witnessed by the following derivation tree:

$$\dfrac{\dfrac{\dfrac{}{\texttt{s} \longrightarrow \texttt{false}}\ \textsc{E-IfTrue}}{\texttt{t} \longrightarrow \texttt{u}}\ \textsc{E-If}}{\texttt{if t then false else false} \longrightarrow \texttt{if u then false else false}}\ \textsc{E-If}$$

- where

$$\texttt{s} \stackrel{\text{def}}{=} \texttt{if true then false else false}$$

$$\texttt{t} \stackrel{\text{def}}{=} \texttt{if s then true else true}$$

$$\texttt{u} \stackrel{\text{def}}{=} \texttt{if false then true else true}$$

# Induction on Derivation

Theorem [Determinacy of one-step evaluation]:

If $t \to t'$ and $t \to t''$, then $t' = t''$.

**Proof**. By induction on derivation of $t \to t'$.

If *the last rule* used in the derivation of $t \to t'$ is E-IfTrue, then $t$ has the form

if true then t2 else t3.

It can be shown that there is only one way to reduce such $t$.

……

# Normal Form

- **Definition**: A term *t* is in normal form if *no evaluation rule* applies to it.

- **Theorem**: Every *value* is in normal form.

- **Theorem**: If *t* is in normal form, then *t* is a *value*.
  - Prove by contradiction (then by structural induction).

# Multi-step Evaluation Relation

- **Definition**: The multi-step evaluation relation $\rightarrow*$ is the *reflexive*, *transitive closure* of one-step evaluation.

- **Theorem** [Uniqueness of normal forms]:

  If t $\rightarrow*$ u and t $\rightarrow*$ u′, where u and u′ are both normal forms, then u = u′.

- **Theorem** [Termination of Evaluation]:

  For every term t there is some normal form t′ such that t $\rightarrow*$ t′.

# Big-step Evaluation

$$v \Downarrow v \qquad \text{(B-VALUE)}$$

$$\frac{t_1 \Downarrow \text{true} \qquad t_2 \Downarrow v_2}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \Downarrow v_2} \qquad \text{(B-IFTRUE)}$$

$$\frac{t_1 \Downarrow \text{false} \qquad t_3 \Downarrow v_3}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \Downarrow v_3} \qquad \text{(B-IFFALSE)}$$

$$\frac{t_1 \Downarrow nv_1}{\text{succ } t_1 \Downarrow \text{succ } nv_1} \qquad \text{(B-SUCC)}$$

$$\frac{t_1 \Downarrow 0}{\text{pred } t_1 \Downarrow 0} \qquad \text{(B-PREDZERO)}$$

$$\frac{t_1 \Downarrow \text{succ } nv_1}{\text{pred } t_1 \Downarrow nv_1} \qquad \text{(B-PREDSUCC)}$$

$$\frac{t_1 \Downarrow 0}{\text{iszero } t_1 \Downarrow \text{true}} \qquad \text{(B-ISZEROZERO)}$$

$$\frac{t_1 \Downarrow \text{succ } nv_1}{\text{iszero } t_1 \Downarrow \text{false}} \qquad \text{(B-ISZEROSUCC)}$$

# Extending Evaluation to Numbers

*New syntactic forms*

| | | | |
|---|---|---|---|
| t | ::= | ... | *terms:* |
| | | 0 | *constant zero* |
| | | succ t | *successor* |
| | | pred t | *predecessor* |
| | | iszero t | *zero test* |
| v | ::= | ... | *values:* |
| | | nv | *numeric value* |
| nv | ::= | | *numeric values:* |
| | | 0 | *zero value* |
| | | succ nv | *successor value* |

*New evaluation rules* $\boxed{t \longrightarrow t'}$

$$\frac{t_1 \longrightarrow t_1'}{\text{succ } t_1 \longrightarrow \text{succ } t_1'} \quad \text{(E-SUCC)}$$

$$\text{pred } 0 \longrightarrow 0 \quad \text{(E-PREDZERO)}$$

$$\text{pred (succ } nv_1) \longrightarrow nv_1 \quad \text{(E-PREDSUCC)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{pred } t_1 \longrightarrow \text{pred } t_1'} \quad \text{(E-PRED)}$$

$$\text{iszero } 0 \longrightarrow \text{true} \quad \text{(E-ISZEROZERO)}$$

$$\text{iszero (succ } nv_1) \longrightarrow \text{false} \quad \text{(E-ISZEROSUCC)}$$

$$\frac{t_1 \longrightarrow t_1'}{\text{iszero } t_1 \longrightarrow \text{iszero } t_1'} \quad \text{(E-ISZERO)}$$

# Stuckness

- Definition: A closed term is stuck if it is in *normal form* but *not a value*.

- Examples:
  - succ true
  - succ false
  - if zero then true else false

# Summary

- How to define syntax?
  - Grammar, Inductively, Inference Rules, Generative

- How to define semantics?
  - Operational, Denotational, Axomatic

- How to define evaluation relation (operational semantics)?
  - Small-step/Big-step evaluation relation
  - Normal form
  - Confluence/termination

# Homework

- Do Exercise 3.5.13 & 3.5.16 in Chapter 3.