

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



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



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

```
if false then 0 else 1 (1 = \operatorname{succ} 0)

\rightarrow 1

iszero (pred (succ 0))

\rightarrow \operatorname{true}

succ(succ(succ(0)))

\rightarrow ?
```





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 $t_1 \in T$,

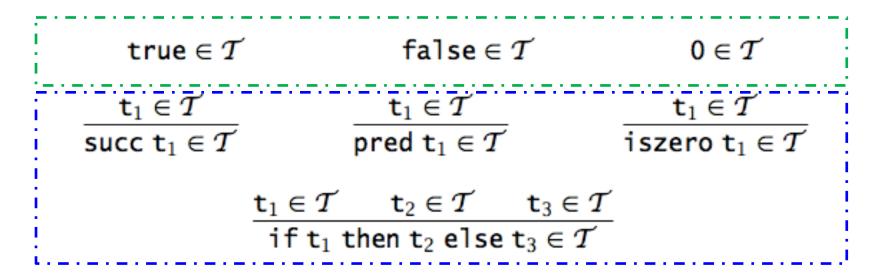
then {succ t_1 , pred t_1 , iszero t_1 } \subseteq T;

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

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



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



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
```



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

$$\begin{array}{rcl} S_0 &=& \varnothing \\ S_{i+1} &=& \{\texttt{true}, \texttt{false}, 0\} \\ && \cup & \{\texttt{succ}\, \texttt{t}_1, \texttt{pred}\, \texttt{t}_1, \texttt{iszero}\, \texttt{t}_1 \mid \texttt{t}_1 \in S_i\} \\ && \cup & \{\texttt{if}\, \texttt{t}_1\, \texttt{then}\, \texttt{t}_2\, \texttt{else}\, \texttt{t}_3 \mid \texttt{t}_1, \texttt{t}_2, \texttt{t}_3 \in S_i\}. \end{array}$$

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



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

- *Consts*(true) {true} *Consts*(false) {false} = Consts(0) {0} = *Consts*(succ t_1) $Consts(t_1)$ = Consts(pred t_1) $Consts(t_1)$ = $Consts(iszero t_1)$ $Consts(t_1)$ = $Consts(if t_1 then t_2 else t_3)$
 - = $Consts(t_1) \cup Consts(t_2) \cup Consts(t_3)$



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

<pre>size(true)</pre>	=	1
<pre>size(false)</pre>	=	1
size(0)	=	1
size(succ t ₁)	=	$size(t_1) + 1$
size(pred t ₁)	=	$size(t_1) + 1$
<i>size</i> (iszerot ₁)	=	$size(t_1) + 1$
size(if t_1 then t_2 else t_3)	=	$size(t_1) + size(t_2) + size(t_3) + 1$



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

- *depth*(true) 1 depth(false) 1 = depth(0) 1 = $depth(succ t_1)$ = $depth(pred t_1)$ = $depth(iszerot_1)$ = $depth(ift_1 then t_2 else t_3)$ =
 - $depth(t_1) + 1$
 - $depth(t_1) + 1$
 - $depth(t_1) + 1$
 - $\max(depth(t_1), depth(t_2), depth(t_3)) + 1$



Lemma.

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

| Consts(t) $| \le size(t)$

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

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

By the induction hypothesis, $|Consts(t1)| \le size(t1)$, and we have : |Consts(t)|

 $= |Consts(t1)| \le size(t1) < size(t).$

– Case t is if t1 then t2 else t3

?



```
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 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



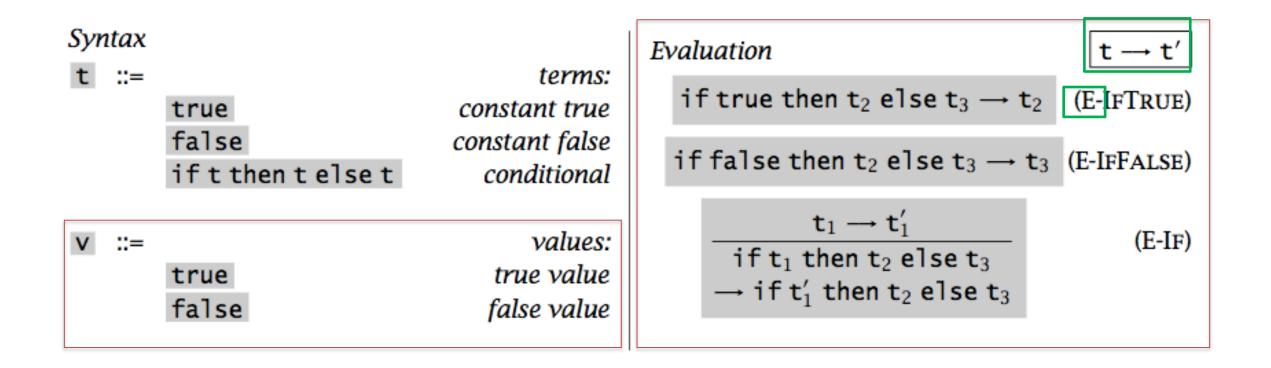
- Axiomatic methods take the *laws* (properties) themselves *as the definition* of the language.
- The meaning of a *term* is just *what* can be proved about it.
 - They focus attention on *the process of reasoning* about programs.
 - Hoare logic: define the meaning of imperative languages



Evaluation

Evaluation relation (small-step/big-step) Normal form Confluence and termination





"t evaluates to t' in one step

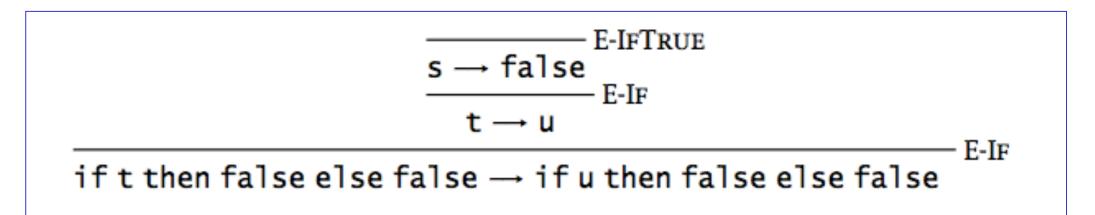


- The *one-step evaluation relation* → is the *smallest binary relation* on terms satisfying the three rules in the previous slide.
- When the *pair* (*t*, *t'*) is in the evaluation relation, we say that "t \rightarrow t' is *derivable*."

Derivation Tree



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



where s ^{def} = if true then false else false
 t ^{def} = if s then true else true
 u ^{def} = if false then true else true



Theorem [Determinacy of one-step evaluation]:

```
If t \rightarrow t' and t \rightarrow t'', then t' = t''.
```

Proof. By induction on derivation of $t \rightarrow t'$.

If *the last rule* used in the derivation of $t \rightarrow t'$ 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*.

••••





- **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).



- Definition: The multi-step evaluation relation →* 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↓v	(B-VALUE)	
$\frac{t_1 \Downarrow true \qquad t_2 \Downarrow v_2}{if t_1 then t_2 else t_2 \Downarrow v_2}$	(B-IFTRUE)	
if t_1 then t_2 else $t_3 \Downarrow v_2$		
$\frac{t_1 \Downarrow false t_3 \Downarrow v_3}{ift_1 then t_2 else t_2 \parallel v_2}$	(B-IFFALSE)	
if t_1 then t_2 else $t_3 \Downarrow v_3$		
$\frac{t_1 \Downarrow n v_1}{succ t_1 \Downarrow succ n v_1}$	(B-SUCC)	
$succ t_1 \Downarrow succ nv_1$		
$\frac{t_1 \Downarrow 0}{nndt_1 \Downarrow 0}$	(B-PredZero)	
pred $t_1 \Downarrow 0$		
$\frac{t_1 \Downarrow succ nv_1}{nnod t}$	(B-PREDSUCC)	
pred $t_1 \Downarrow nv_1$		
$\frac{t_1 \Downarrow 0}{i_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c_{c}}}}}}}}}$	(B-ISZEROZERO)	
iszero t ₁ ↓true		
$t_1 \Downarrow succ nv_1$	(B-ISZEROSUCC)	
iszero t ₁ ↓false		

Extending Evaluation to Numbers

New syntactic forms		New evaluation rules	$t \rightarrow t'$	
t ::= 0 succt	t ::= terms: 0 constant zero succ t successor	$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{succ} \mathtt{t}_1 \to \mathtt{succ} \mathtt{t}_1'}$	(E-Succ)	
pred t iszero t	predecessor zero test	pred $0 \rightarrow 0$	(E-PredZero)	
v ::=	values:	pred (succ nv_1) $\rightarrow nv_1$	(E-PREDSUCC)	
nv nv ::=	numeric value numeric values:	$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\texttt{pred} \mathtt{t}_1 \longrightarrow \texttt{pred} \mathtt{t}_1'}$	(E-Pred)	
0 SUCC DV	0 zero value succ ny successor value	$\begin{array}{ccc} 0 & zero \ value \\ succ \ nv & successor \ value \end{array} & is zero \ 0 \rightarrow \end{array}$	iszero 0 \rightarrow true	(E-ISZEROZERO)
Successor Func		iszero (succ nv ₁) \rightarrow fals	e (E-IszeroSucc)	
		$\frac{\texttt{t}_1 \rightarrow \texttt{t}_1'}{\texttt{iszero}\texttt{t}_1 \rightarrow \texttt{iszero}\texttt{t}_1'}$	(E-IsZero)	





- 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





- 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





• Do Exercise 3.5.13 & 3.5.16 in Chapter 3.