



# *CS618: Program Analysis*

## *2016-17 1<sup>st</sup> Semester*

## *Types and Program Analysis*

Amey Karkare

karkare@cse.iitk.ac.in  
karkare@cse.iitb.ac.in

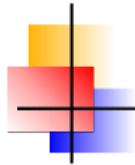
Department of CSE, IIT Kanpur/Bombay





## *Reference Book*

Types and Programming Languages by Benjamin C. Pierce



## Type: Definition

### type

/tʌɪp/ ⓘ

*noun*

1. a category of people or things having common characteristics.

"this type of heather grows better in a drier habitat"

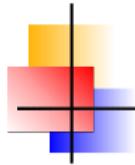
*synonyms:* kind, sort, variety, class, category, classification, group, set, bracket, genre, genus, species, family, order, breed, race, strain; More

2. a person or thing exemplifying the ideal or defining characteristics of something.

"she characterized his witty sayings as the type of modern wisdom"

*synonyms:* epitome, quintessence, essence, perfect example, archetype, model, pattern, paradigm, exemplar, embodiment, personification, avatar; prototype

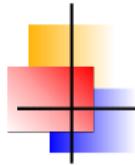
"she characterized his witty sayings as the type of modern wisdom"



## *Types in Programming*

- ▶ A collection of *values*





## Types in Programming

- ▶ A collection of *values*



- ▶ The operations that are permitted on these values



## Type System

- ▶ A collection of rules for checking the correctness of usages of types



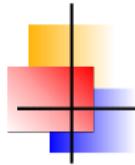
## Type System

- ▶ A collection of rules for checking the correctness of usages of types
  - ▶ “Consistency” of programs



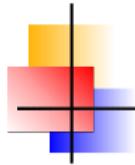
# *The World of Programming Languages*

- ▶ Typed



## *The World of Programming Languages*

- ▶ Typed
  - ▶ C, C++, Java, Python, ...



## *The World of Programming Languages*

- ▶ Typed
  - ▶ C, C++, Java, Python, ...
- ▶ Untyped



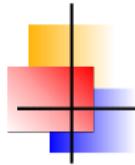
## The World of Programming Languages

- ▶ Typed
  - ▶ C, C++, Java, Python, ...
- ▶ Untyped
  - ▶ Assembly, *any other?*



# *The World of Programming Languages*

	<b>Statically Typed</b>	<b>Dynamically Typed</b>
<b>Strongly Typed</b>		
<b>Weakly Typed</b>		



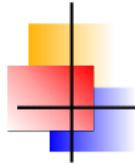
# *The World of Programming Languages*

	<b>Statically Typed</b>	<b>Dynamically Typed</b>
<b>Strongly Typed</b>	ML, Haskell, Pascal (almost), Java (almost)	
<b>Weakly Typed</b>		



# The World of Programming Languages

	<b>Statically Typed</b>	<b>Dynamically Typed</b>
<b>Strongly Typed</b>	ML, Haskell, Pascal (almost), Java (almost)	Lisp, Scheme
<b>Weakly Typed</b>		



# The World of Programming Languages

	<b>Statically Typed</b>	<b>Dynamically Typed</b>
<b>Strongly Typed</b>	ML, Haskell, Pascal (almost), Java (almost)	Lisp, Scheme
<b>Weakly Typed</b>	C, C++	



# The World of Programming Languages

	<b>Statically Typed</b>	<b>Dynamically Typed</b>
<b>Strongly Typed</b>	ML, Haskell, Pascal (almost), Java (almost)	Lisp, Scheme
<b>Weakly Typed</b>	C, C++	Perl



## *Applications of Type-based Analyses*

- ▶ Error Detection



## *Applications of Type-based Analyses*

- ▶ Error Detection
  - ▶ Language Safety



## *Applications of Type-based Analyses*

- ▶ Error Detection
  - ▶ Language Safety
  - ▶ Verification



## *Applications of Type-based Analyses*

- ▶ Error Detection
  - ▶ Language Safety
  - ▶ Verification
- ▶ Abstraction



## *Applications of Type-based Analyses*

- ▶ Error Detection
  - ▶ Language Safety
  - ▶ Verification
- ▶ Abstraction
- ▶ Documentation



## *Applications of Type-based Analyses*

- ▶ Error Detection
  - ▶ Language Safety
  - ▶ Verification
- ▶ Abstraction
- ▶ Documentation
- ▶ Maintenance



## *Applications of Type-based Analyses*

- ▶ Error Detection
  - ▶ Language Safety
  - ▶ Verification
- ▶ Abstraction
- ▶ Documentation
- ▶ Maintenance
- ▶ Efficiency



## Untyped Arithmetic Expression Language

$t :=$  – *terms*



## Untyped Arithmetic Expression Language

$t :=$

true

- *terms*
- *constant true*



## Untyped Arithmetic Expression Language

$t :=$	$- \text{ terms}$
true	$- \text{ constant true}$
false	$- \text{ constant false}$



## Untyped Arithmetic Expression Language

$t :=$  – *terms*  
    true                          – *constant true*  
    false                         – *constant false*  
    if  $t$  then  $t$  else  $t$     – *conditional*



## Untyped Arithmetic Expression Language

$t :=$	– <i>terms</i>
true	– <i>constant true</i>
false	– <i>constant false</i>
if $t$ then $t$ else $t$	– <i>conditional</i>
0	– <i>constant zero</i>



## Untyped Arithmetic Expression Language

$t :=$	– <i>terms</i>
true	– <i>constant true</i>
false	– <i>constant false</i>
if $t$ then $t$ else $t$	– <i>conditional</i>
0	– <i>constant zero</i>
succ $t$	– <i>successor</i>



## Untyped Arithmetic Expression Language

$t :=$	– <i>terms</i>
true	– <i>constant true</i>
false	– <i>constant false</i>
if $t$ then $t$ else $t$	– <i>conditional</i>
0	– <i>constant zero</i>
succ $t$	– <i>successor</i>
pred $t$	– <i>predecessor</i>



## Untyped Arithmetic Expression Language

$t :=$	– <i>terms</i>
true	– <i>constant true</i>
false	– <i>constant false</i>
if $t$ then $t$ else $t$	– <i>conditional</i>
0	– <i>constant zero</i>
succ $t$	– <i>successor</i>
pred $t$	– <i>predecessor</i>
iszero $t$	– <i>zero test</i>



## Syntax: Inductive Definition

The set of *terms* is the smallest set  $\mathcal{T}$  such that



## Syntax: Inductive Definition

The set of *terms* is the smallest set  $\mathcal{T}$  such that

1.  $\{\text{true}, \text{false}, 0\} \subseteq \mathcal{T}$



## Syntax: Inductive Definition

The set of *terms* is the smallest set  $\mathcal{T}$  such that

1.  $\{\text{true}, \text{false}, 0\} \subseteq \mathcal{T}$
2. if  $t_1 \in \mathcal{T}$ , then  $\{\text{succ } t_1, \text{pred } t_1, \text{iszero } t_1\} \subseteq \mathcal{T}$



## Syntax: Inductive Definition

The set of *terms* is the smallest set  $\mathcal{T}$  such that

1.  $\{\text{true}, \text{false}, 0\} \subseteq \mathcal{T}$
2. if  $t_1 \in \mathcal{T}$ , then  $\{\text{succ } t_1, \text{pred } t_1, \text{iszero } t_1\} \subseteq \mathcal{T}$
3. if  $t_1 \in \mathcal{T}$ ,  $t_2 \in \mathcal{T}$ , and  $t_3 \in \mathcal{T}$  then  $\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \in \mathcal{T}$



## Syntax: Inference Rules

The set of *terms*,  $\mathcal{T}$  is defined by the following rules:



## Syntax: Inference Rules

The set of *terms*,  $\mathcal{T}$  is defined by the following rules:

$$\text{true} \in \mathcal{T}$$

$$\text{false} \in \mathcal{T}$$

$$0 \in \mathcal{T}$$



## Syntax: Inference Rules

The set of *terms*,  $\mathcal{T}$  is defined by the following rules:

$$\text{true} \in \mathcal{T}$$

$$\text{false} \in \mathcal{T}$$

$$0 \in \mathcal{T}$$

$$\frac{t_1 \in \mathcal{T}}{\text{succ } t_1 \in \mathcal{T}}$$

$$\frac{t_1 \in \mathcal{T}}{\text{pred } t_1 \in \mathcal{T}}$$

$$\frac{t_1 \in \mathcal{T}}{\text{iszero } t_1 \in \mathcal{T}}$$



## Syntax: Inference Rules

The set of *terms*,  $\mathcal{T}$  is defined by the following rules:

$$\text{true} \in \mathcal{T}$$

$$\text{false} \in \mathcal{T}$$

$$0 \in \mathcal{T}$$

$$\frac{t_1 \in \mathcal{T}}{\text{succ } t_1 \in \mathcal{T}}$$

$$\frac{t_1 \in \mathcal{T}}{\text{pred } t_1 \in \mathcal{T}}$$

$$\frac{t_1 \in \mathcal{T}}{\text{iszero } t_1 \in \mathcal{T}}$$

$$\frac{t_1 \in \mathcal{T} \quad t_2 \in \mathcal{T} \quad t_3 \in \mathcal{T}}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \in \mathcal{T}}$$



## *Concrete Syntax*

$$\mathcal{S}_0 = \emptyset$$



## Concrete Syntax

$$\mathcal{S}_0 = \emptyset$$

$$\mathcal{S}_{i+1} = \{\text{true}, \text{false}, 0\}$$



## Concrete Syntax

$$\mathcal{S}_0 = \emptyset$$

$$\begin{aligned}\mathcal{S}_{i+1} = & \{\text{true}, \text{false}, 0\} \\ & \cup \{\text{succ } t_1, \text{pred } t_1, \text{iszero } t_1 \mid t_1 \in \mathcal{S}_i\}\end{aligned}$$



## Concrete Syntax

$$\mathcal{S}_0 = \emptyset$$

$$\begin{aligned}\mathcal{S}_{i+1} = & \{ \text{true}, \text{false}, 0 \} \\ & \cup \{ \text{succ } t_1, \text{pred } t_1, \text{iszero } t_1 \mid t_1 \in \mathcal{S}_i \} \\ & \cup \{ \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid t_1, t_2, t_3 \in \mathcal{S}_i \}\end{aligned}$$



## Concrete Syntax

$$\mathcal{S}_0 = \emptyset$$

$$\begin{aligned}\mathcal{S}_{i+1} = & \{\text{true}, \text{false}, 0\} \\ & \cup \{\text{succ } t_1, \text{pred } t_1, \text{iszero } t_1 \mid t_1 \in \mathcal{S}_i\} \\ & \cup \{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid t_1, t_2, t_3 \in \mathcal{S}_i\}\end{aligned}$$

Let  $\mathcal{S} = \bigcup_i \mathcal{S}_i$ .



## Concrete Syntax

$$\mathcal{S}_0 = \emptyset$$

$$\begin{aligned}\mathcal{S}_{i+1} = & \{\text{true}, \text{false}, 0\} \\ & \cup \{\text{succ } t_1, \text{pred } t_1, \text{iszero } t_1 \mid t_1 \in \mathcal{S}_i\} \\ & \cup \{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid t_1, t_2, t_3 \in \mathcal{S}_i\}\end{aligned}$$

Let  $\mathcal{S} = \bigcup_i \mathcal{S}_i$ .

Then,  $\mathcal{T} = \mathcal{S}$ .



## *Induction on Terms*

- ▶ Any  $t \in \mathcal{T}$



## *Induction on Terms*

- ▶ Any  $t \in \mathcal{T}$ 
  - ▶ Either a ground term, i.e.  $\in \{\text{true}, \text{false}, 0\}$



## Induction on Terms

- ▶ Any  $t \in \mathcal{T}$ 
  - ▶ Either a ground term, i.e.  $\in \{\text{true}, \text{false}, 0\}$
  - ▶ Or is created from some smaller terms  $\in \mathcal{T}$



## Induction on Terms

- ▶ Any  $t \in \mathcal{T}$ 
  - ▶ Either a ground term, i.e.  $\in \{\text{true}, \text{false}, 0\}$
  - ▶ Or is created from some smaller terms  $\in \mathcal{T}$
- ▶ Allows for inductive definitions and inductive proofs.



## Induction on Terms

- ▶ Any  $t \in \mathcal{T}$ 
  - ▶ Either a ground term, i.e.  $\in \{\text{true}, \text{false}, 0\}$
  - ▶ Or is created from some smaller terms  $\in \mathcal{T}$
- ▶ Allows for inductive definitions and inductive proofs.
- ▶ Three sample inductive properties



## Induction on Terms

- ▶ Any  $t \in \mathcal{T}$ 
  - ▶ Either a ground term, i.e.  $\in \{\text{true}, \text{false}, 0\}$
  - ▶ Or is created from some smaller terms  $\in \mathcal{T}$
- ▶ Allows for inductive definitions and inductive proofs.
- ▶ Three sample inductive properties
  - ▶  $\text{Consts}(t)$



## Induction on Terms

- ▶ Any  $t \in \mathcal{T}$ 
  - ▶ Either a ground term, i.e.  $\in \{\text{true}, \text{false}, 0\}$
  - ▶ Or is created from some smaller terms  $\in \mathcal{T}$
- ▶ Allows for inductive definitions and inductive proofs.
- ▶ Three sample inductive properties
  - ▶  $\text{Consts}(t)$
  - ▶  $\text{size}(t)$



## Induction on Terms

- ▶ Any  $t \in \mathcal{T}$ 
  - ▶ Either a ground term, i.e.  $\in \{\text{true}, \text{false}, 0\}$
  - ▶ Or is created from some smaller terms  $\in \mathcal{T}$
- ▶ Allows for inductive definitions and inductive proofs.
- ▶ Three sample inductive properties
  - ▶  $\text{Consts}(t)$
  - ▶  $\text{size}(t)$
  - ▶  $\text{depth}(t)$



## Consts

- ▶ The set of constants in a term t.



## Consts

- ▶ The set of constants in a term t.

$$\textit{Consts}(\text{true}) = \{\text{true}\}$$

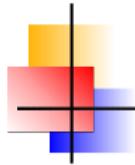


## Consts

- ▶ The set of constants in a term t.

$$\textit{Consts}(\text{true}) = \{\text{true}\}$$

$$\textit{Consts}(\text{false}) = \{\text{false}\}$$



## Consts

- ▶ The set of constants in a term t.

$$\textit{Consts}(\text{true}) = \{\text{true}\}$$

$$\textit{Consts}(\text{false}) = \{\text{false}\}$$

$$\textit{Consts}(0) = \{0\}$$

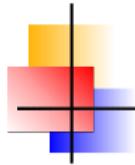
- ▶ The set of constants in a term t.

$$\textit{Consts}(\text{true}) = \{\text{true}\}$$

$$\textit{Consts}(\text{false}) = \{\text{false}\}$$

$$\textit{Consts}(0) = \{0\}$$

$$\textit{Consts}(\text{succ } t) = \textit{Consts}(t)$$



## Consts

- ▶ The set of constants in a term t.

$$\text{Consts}(\text{true}) = \{\text{true}\}$$

$$\text{Consts}(\text{false}) = \{\text{false}\}$$

$$\text{Consts}(0) = \{0\}$$

$$\text{Consts}(\text{succ } t) = \text{Consts}(t)$$

$$\text{Consts}(\text{pred } t) = \text{Consts}(t)$$



## Consts

- ▶ The set of constants in a term t.

$$\text{Consts}(\text{true}) = \{\text{true}\}$$

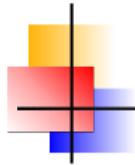
$$\text{Consts}(\text{false}) = \{\text{false}\}$$

$$\text{Consts}(0) = \{0\}$$

$$\text{Consts}(\text{succ } t) = \text{Consts}(t)$$

$$\text{Consts}(\text{pred } t) = \text{Consts}(t)$$

$$\text{Consts}(\text{iszero } t) = \text{Consts}(t)$$



## Consts

- ▶ The set of constants in a term t.

$$\text{Consts}(\text{true}) = \{\text{true}\}$$

$$\text{Consts}(\text{false}) = \{\text{false}\}$$

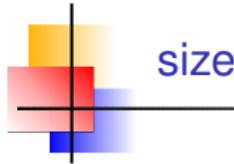
$$\text{Consts}(0) = \{0\}$$

$$\text{Consts}(\text{succ } t) = \text{Consts}(t)$$

$$\text{Consts}(\text{pred } t) = \text{Consts}(t)$$

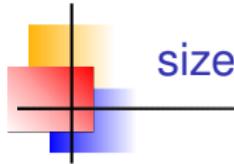
$$\text{Consts}(\text{iszero } t) = \text{Consts}(t)$$

$$\begin{aligned}\text{Consts}(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) &= \text{Consts}(t_1) \\ &\quad \cup \text{Consts}(t_2) \\ &\quad \cup \text{Consts}(t_3)\end{aligned}$$



size

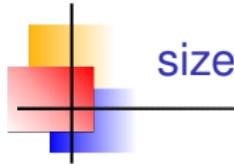
- ▶ The number of nodes in the abstract syntax tree of a term  $t$ .



size

- ▶ The number of nodes in the abstract syntax tree of a term  $t$ .

$$\text{size}(\text{true}) = 1$$

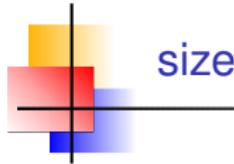


size

- ▶ The number of nodes in the abstract syntax tree of a term  $t$ .

$$\text{size}(\text{true}) = 1$$

$$\text{size}(\text{false}) = 1$$



size

- ▶ The number of nodes in the abstract syntax tree of a term  $t$ .

$$\text{size}(\text{true}) = 1$$

$$\text{size}(\text{false}) = 1$$

$$\text{size}(0) = 1$$



## size

- ▶ The number of nodes in the abstract syntax tree of a term  $t$ .

$$\text{size}(\text{true}) = 1$$

$$\text{size}(\text{false}) = 1$$

$$\text{size}(0) = 1$$

$$\text{size}(\text{succ } t) = \text{size}(t) + 1$$



## size

- ▶ The number of nodes in the abstract syntax tree of a term  $t$ .

$$\text{size}(\text{true}) = 1$$

$$\text{size}(\text{false}) = 1$$

$$\text{size}(0) = 1$$

$$\text{size}(\text{succ } t) = \text{size}(t) + 1$$

$$\text{size}(\text{pred } t) = \text{size}(t) + 1$$



## size

- ▶ The number of nodes in the abstract syntax tree of a term  $t$ .

$$\text{size}(\text{true}) = 1$$

$$\text{size}(\text{false}) = 1$$

$$\text{size}(0) = 1$$

$$\text{size}(\text{succ } t) = \text{size}(t) + 1$$

$$\text{size}(\text{pred } t) = \text{size}(t) + 1$$

$$\text{size}(\text{iszero } t) = \text{size}(t) + 1$$



## size

- ▶ The number of nodes in the abstract syntax tree of a term  $t$ .

$$\text{size}(\text{true}) = 1$$

$$\text{size}(\text{false}) = 1$$

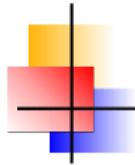
$$\text{size}(0) = 1$$

$$\text{size}(\text{succ } t) = \text{size}(t) + 1$$

$$\text{size}(\text{pred } t) = \text{size}(t) + 1$$

$$\text{size}(\text{iszero } t) = \text{size}(t) + 1$$

$$\text{size}(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) = \text{size}(t_1) + \text{size}(t_2) + \text{size}(t_3)$$



## depth

- ▶ The maximum depth of the abstract syntax tree of a term  $t$ .
- ▶ Equivalently, the smallest  $i$  such that  $t \in S_i$ .



## depth

- ▶ The maximum depth of the abstract syntax tree of a term t.
- ▶ Equivalently, the smallest  $i$  such that  $t \in S_i$ .

$$\text{depth}(\text{true}) = 1$$



## depth

- ▶ The maximum depth of the abstract syntax tree of a term  $t$ .
- ▶ Equivalently, the smallest  $i$  such that  $t \in S_i$ .

$$\text{depth}(\text{true}) = 1$$

$$\text{depth}(\text{false}) = 1$$



## depth

- ▶ The maximum depth of the abstract syntax tree of a term  $t$ .
- ▶ Equivalently, the smallest  $i$  such that  $t \in S_i$ .

$$\text{depth}(\text{true}) = 1$$

$$\text{depth}(\text{false}) = 1$$

$$\text{depth}(0) = 1$$



## depth

- ▶ The maximum depth of the abstract syntax tree of a term t.
- ▶ Equivalently, the smallest  $i$  such that  $t \in S_i$ .

$$\text{depth}(\text{true}) = 1$$

$$\text{depth}(\text{false}) = 1$$

$$\text{depth}(0) = 1$$

$$\text{depth}(\text{succ } t) = \text{depth}(t) + 1$$



## depth

- ▶ The maximum depth of the abstract syntax tree of a term  $t$ .
- ▶ Equivalently, the smallest  $i$  such that  $t \in S_i$ .

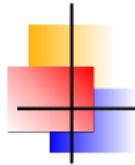
$$\text{depth}(\text{true}) = 1$$

$$\text{depth}(\text{false}) = 1$$

$$\text{depth}(0) = 1$$

$$\text{depth}(\text{succ } t) = \text{depth}(t) + 1$$

$$\text{depth}(\text{pred } t) = \text{depth}(t) + 1$$



## depth

- ▶ The maximum depth of the abstract syntax tree of a term t.
- ▶ Equivalently, the smallest  $i$  such that  $t \in S_i$ .

$$\text{depth}(\text{true}) = 1$$

$$\text{depth}(\text{false}) = 1$$

$$\text{depth}(0) = 1$$

$$\text{depth}(\text{succ } t) = \text{depth}(t) + 1$$

$$\text{depth}(\text{pred } t) = \text{depth}(t) + 1$$

$$\text{depth}(\text{iszero } t) = \text{depth}(t) + 1$$



## depth

- ▶ The maximum depth of the abstract syntax tree of a term  $t$ .
- ▶ Equivalently, the smallest  $i$  such that  $t \in S_i$ .

$$\text{depth}(\text{true}) = 1$$

$$\text{depth}(\text{false}) = 1$$

$$\text{depth}(0) = 1$$

$$\text{depth}(\text{succ } t) = \text{depth}(t) + 1$$

$$\text{depth}(\text{pred } t) = \text{depth}(t) + 1$$

$$\text{depth}(\text{iszero } t) = \text{depth}(t) + 1$$

$$\begin{aligned}\text{depth}(\text{if } t_1 \text{ then } t_2 \text{ else } t_3) &= \max(\text{depth}(t_1) + \text{depth}(t_2) \\ &\quad + \text{depth}(t_3)) + 1\end{aligned}$$



## *A Simple Property of Terms*

- ▶ The number of distinct constants in a term  $t$  is no greater than the size of  $t$ .

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

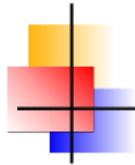


## *A Simple Property of Terms*

- ▶ The number of distinct constants in a term  $t$  is no greater than the size of  $t$ .

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

- ▶ **Proof:** Exercise.



## *The Set of Values*

V :=

– *values*



## The Set of Values

$V :=$                    – *values*  
    true                   – *value true*



## The Set of Values

V :=

true

false

– *values*

– *value true*

– *value false*



## The Set of Values

V :=

true

false

0

– *values*

– *value true*

– *value false*

– *value zero*



## The Set of Values

$V :=$	$- values$
true	$- value true$
false	$- value false$
0	$- value zero$
succ $V$	$- successor value$



## *Small-step Operational Semantics*

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”



## *Small-step Operational Semantics*

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

if true then  $t_2$  else  $t_3 \rightarrow t_2$



## *Small-step Operational Semantics*

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

if true then  $t_2$  else  $t_3 \rightarrow t_2$

if false then  $t_2$  else  $t_3 \rightarrow t_3$



## Small-step Operational Semantics

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

if true then  $t_2$  else  $t_3 \rightarrow t_2$

if false then  $t_2$  else  $t_3 \rightarrow t_3$

$$\frac{t_1 \rightarrow t'_1}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t'_1 \text{ then } t_2 \text{ else } t_3}$$

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

$$\frac{t_1 \rightarrow t'_1}{\text{succ } t_1 \rightarrow \text{succ } t'_1}$$



## *Small-step Operational Semantics (contd...)*

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

$$\frac{t_1 \rightarrow t'_1}{\text{succ } t_1 \rightarrow \text{succ } t'_1}$$

pred  $0 \rightarrow 0$



## Small-step Operational Semantics (contd...)

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

$$\frac{t_1 \rightarrow t'_1}{\text{succ } t_1 \rightarrow \text{succ } t'_1}$$

$$\text{pred } 0 \rightarrow 0$$

$$\text{pred } (\text{succ } v) \rightarrow v$$



## Small-step Operational Semantics (contd...)

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

$$\frac{t_1 \rightarrow t'_1}{\text{succ } t_1 \rightarrow \text{succ } t'_1}$$

$$\text{pred } 0 \rightarrow 0$$

$$\text{pred } (\text{succ } v) \rightarrow v$$

$$\frac{t_1 \rightarrow t'_1}{\text{pred } t_1 \rightarrow \text{pred } t'_1}$$



## *Small-step Operational Semantics (contd...)*

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

iszero  $0 \rightarrow \text{true}$



## *Small-step Operational Semantics (contd...)*

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

iszero 0 → true

iszero (succ v) → false



## Small-step Operational Semantics (contd...)

- ▶  $t \rightarrow t'$  denotes “ $t$  evaluates to  $t'$  in one step”

$$\text{iszzero } 0 \rightarrow \text{true}$$

$$\text{iszzero } (\text{succ } v) \rightarrow \text{false}$$

$$\frac{t_1 \rightarrow t'_1}{\text{iszzero } t_1 \rightarrow \text{iszzero } t'_1}$$



## *Normal Form*

- ▶ A term is  $t$  in normal form if no evaluation rule applies to it.



## *Normal Form*

- ▶ A term is  $t$  in normal form if no evaluation rule applies to it.
- ▶ In other words, there is no  $t'$  such that  $t \rightarrow t'$ .



## Evaluation Sequence

- ▶ An evaluation sequence starting from a term  $t$  is a (finite or infinite) sequence of terms  $t_1, t_2, \dots$ , such that

$$t \rightarrow t_1$$

$$t_1 \rightarrow t_2$$

etc.



## *Stuck Term*

- ▶ A term is said to be **stuck** if it is a normal form but not a value.



## *Stuck Term*

- ▶ A term is said to be **stuck** if it is a normal form but not a value.
- ▶ A simple notion of “run-time type error”