CS738: Advanced Compiler Optimizations

Types and Program Analysis

Amey Karkare
karkare@cse.iitk.ac.in

http://www.cse.iitk.ac.in/~karkare/cs738
Department of CSE, IIT Kanpur

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

	Statically Typed	Dynamically Typed
Strongly Typed		
Weekly Typed		

The World of Programming Languages

	Statically Typed	**Dynamically Typed**
Strongly Typed	ML, Haskell, Pascal (almost), Java (almost)	
Weekly Typed		

The World of Programming Languages

	Statically Typed	**Dynamically Typed**
Strongly Typed	ML, Haskell, Pascal (almost), Java (almost)	Lisp, Scheme
Weekly Typed		

The World of Programming Languages

	Statically Typed	**Dynamically Typed**
Strongly Typed	ML, Haskell, Pascal (almost), Java (almost)	Lisp, Scheme
Weekly Typed	C, C++	

The World of Programming Languages

	Statically Typed	**Dynamically Typed**
Strongly Typed	ML, Haskell, Pascal (almost), Java (almost)	Lisp, Scheme
Weekly Typed	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

$\mathrm{t} :=$ – *terms*

Untyped Arithmetic Expression Language

t :=	– *terms*
`true`	– *constant true*

Untyped Arithmetic Expression Language

t :=	– *terms*
`true`	– *constant true*
`false`	– *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

$\mathrm{t} :=$	– *terms*
`true`	– *constant true*
`false`	– *constant false*
`if` t `then` t `else` t	– *conditional*
0	– *constant zero*

Untyped Arithmetic Expression Language

$\mathrm{t} :=$	– *terms*
`true`	– *constant true*
`false`	– *constant false*
`if` t `then` t `else` t	– *conditional*
0	– *constant zero*
`succ` t	– *successor*

Untyped Arithmetic Expression Language

$\mathrm{t} :=$	– *terms*
`true`	– *constant true*
`false`	– *constant false*
`if` t `then` t `else` t	– *conditional*
0	– *constant zero*
`succ` t	– *successor*
`pred` t	– *predecessor*

Untyped Arithmetic Expression Language

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*

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

Syntax: Inductive Definition

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

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

Syntax: Inductive Definition

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

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

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:

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

Syntax: Inference Rules

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

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

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

Syntax: Inference Rules

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

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

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

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

Concrete Syntax

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

Concrete Syntax

$$
\begin{array}{rcl}
\mathcal{S}_0 & = & \emptyset \\
\mathcal{S}_{i+1} & = & \quad \{\mathtt{true}, \mathtt{false}, 0\}
\end{array}
$$

Concrete Syntax

$$
\begin{array}{rcl}
\mathcal{S}_0 & = & \emptyset \\
\mathcal{S}_{i+1} & = & \quad \{\texttt{true}, \texttt{false}, 0\} \\
 & & \cup \, \{\texttt{succ}\ \mathsf{t}_1, \texttt{pred}\ \mathsf{t}_1, \texttt{iszero}\ \mathsf{t}_1 \mid \mathsf{t}_1 \in \mathcal{S}_i\}
\end{array}
$$

Concrete Syntax

$$
\begin{array}{rcl}
\mathcal{S}_0 & = & \emptyset \\
\mathcal{S}_{i+1} & = & \quad \{\texttt{true}, \texttt{false}, 0\} \\
 & & \cup \, \{\texttt{succ}\ t_1, \texttt{pred}\ t_1, \texttt{iszero}\ t_1 \mid t_1 \in \mathcal{S}_i\} \\
 & & \cup \, \{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \mid t_1, t_2, t_2 \in \mathcal{S}_i\}
\end{array}
$$

Concrete Syntax

$$
\begin{array}{rcl}
\mathcal{S}_0 & = & \emptyset \\
\mathcal{S}_{i+1} & = & \quad \{\texttt{true}, \texttt{false}, 0\} \\
& & \cup \; \{\texttt{succ}\ t_1, \texttt{pred}\ t_1, \texttt{iszero}\ t_1 \mid t_1 \in \mathcal{S}_i\} \\
& & \cup \; \{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \mid t_1, t_2, t_2 \in \mathcal{S}_i\}
\end{array}
$$

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

Concrete Syntax

$$
\begin{array}{rcl}
\mathcal{S}_0 & = & \emptyset \\
\mathcal{S}_{i+1} & = & \quad \{\texttt{true}, \texttt{false}, 0\} \\
& & \cup \, \{\texttt{succ}\ t_1, \texttt{pred}\ t_1, \texttt{iszero}\ t_1 \mid t_1 \in \mathcal{S}_i\} \\
& & \cup \, \{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \mid t_1, t_2, t_2 \in \mathcal{S}_i\}
\end{array}
$$

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

Induction on Terms

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

Induction on Terms

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

Induction on Terms

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

Induction on Terms

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

Induction on Terms

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

Consts

- The set of constants in a term t.

Consts

- The set of constants in a term t.

$$\textit{Consts}(\texttt{true}) \quad = \quad \{\texttt{true}\}$$

Consts

- The set of constants in a term t.

$$
\begin{aligned}
\mathit{Consts}(\texttt{true}) &= \{\texttt{true}\} \\
\mathit{Consts}(\texttt{false}) &= \{\texttt{false}\}
\end{aligned}
$$

Consts

- The set of constants in a term t.

$$
\begin{aligned}
\mathit{Consts}(\texttt{true}) &= \{\texttt{true}\}\\
\mathit{Consts}(\texttt{false}) &= \{\texttt{false}\}\\
\mathit{Consts}(0) &= \{0\}
\end{aligned}
$$

Consts

- The set of constants in a term t.

$$
\begin{aligned}
\mathit{Consts}(\texttt{true}) &= \{\texttt{true}\} \\
\mathit{Consts}(\texttt{false}) &= \{\texttt{false}\} \\
\mathit{Consts}(0) &= \{0\} \\
\mathit{Consts}(\texttt{succ}\ \mathrm{t}) &= \mathit{Consts}(\mathrm{t})
\end{aligned}
$$

Consts

- The set of constants in a term t.

$$
\begin{array}{rcl}
\mathit{Consts}(\texttt{true}) & = & \{\texttt{true}\} \\
\mathit{Consts}(\texttt{false}) & = & \{\texttt{false}\} \\
\mathit{Consts}(0) & = & \{0\} \\
\mathit{Consts}(\texttt{succ}\ \mathrm{t}) & = & \mathit{Consts}(\mathrm{t}) \\
\mathit{Consts}(\texttt{pred}\ \mathrm{t}) & = & \mathit{Consts}(\mathrm{t})
\end{array}
$$

Consts

- The set of constants in a term t.

$$
\begin{array}{rcl}
\mathit{Consts}(\mathtt{true}) & = & \{\mathtt{true}\} \\
\mathit{Consts}(\mathtt{false}) & = & \{\mathtt{false}\} \\
\mathit{Consts}(0) & = & \{0\} \\
\mathit{Consts}(\mathtt{succ}\ \mathrm{t}) & = & \mathit{Consts}(\mathrm{t}) \\
\mathit{Consts}(\mathtt{pred}\ \mathrm{t}) & = & \mathit{Consts}(\mathrm{t}) \\
\mathit{Consts}(\mathtt{iszero}\ \mathrm{t}) & = & \mathit{Consts}(\mathrm{t})
\end{array}
$$

Consts

- The set of constants in a term t.

$$
\begin{array}{rcl}
\mathit{Consts}(\texttt{true}) & = & \{\texttt{true}\} \\
\mathit{Consts}(\texttt{false}) & = & \{\texttt{false}\} \\
\mathit{Consts}(0) & = & \{0\} \\
\mathit{Consts}(\texttt{succ}\ \mathrm{t}) & = & \mathit{Consts}(\mathrm{t}) \\
\mathit{Consts}(\texttt{pred}\ \mathrm{t}) & = & \mathit{Consts}(\mathrm{t}) \\
\mathit{Consts}(\texttt{iszero}\ \mathrm{t}) & = & \mathit{Consts}(\mathrm{t}) \\
\mathit{Consts}(\texttt{if}\ \mathrm{t}_1\ \texttt{then}\ \mathrm{t}_2\ \texttt{else}\ \mathrm{t}_3) & = & \mathit{Consts}(\mathrm{t}_1) \\
 & & \cup\ \mathit{Consts}(\mathrm{t}_2) \\
 & & \cup\ \mathit{Consts}(\mathrm{t}_3)
\end{array}
$$

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.

$$\mathit{size}(\texttt{true}) = 1$$

size

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

$$
\begin{aligned}
\mathit{size}(\texttt{true}) &= 1 \\
\mathit{size}(\texttt{false}) &= 1
\end{aligned}
$$

size

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

$$\begin{aligned} \mathit{size}(\texttt{true}) &= 1 \\ \mathit{size}(\texttt{false}) &= 1 \\ \mathit{size}(0) &= 1 \end{aligned}$$

size

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

$$
\begin{aligned}
\mathit{size}(\mathtt{true}) &= 1 \\
\mathit{size}(\mathtt{false}) &= 1 \\
\mathit{size}(0) &= 1 \\
\mathit{size}(\mathtt{succ}\ \mathrm{t}) &= \mathit{size}(\mathrm{t}) + 1
\end{aligned}
$$

size

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

$$
\begin{aligned}
\textit{size}(\texttt{true}) &= 1 \\
\textit{size}(\texttt{false}) &= 1 \\
\textit{size}(0) &= 1 \\
\textit{size}(\texttt{succ}\ \mathrm{t}) &= \textit{size}(\mathrm{t}) + 1 \\
\textit{size}(\texttt{pred}\ \mathrm{t}) &= \textit{size}(\mathrm{t}) + 1
\end{aligned}
$$

size

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

$$
\begin{aligned}
\mathit{size}(\texttt{true}) &= 1 \\
\mathit{size}(\texttt{false}) &= 1 \\
\mathit{size}(0) &= 1 \\
\mathit{size}(\texttt{succ}\ \mathrm{t}) &= \mathit{size}(\mathrm{t}) + 1 \\
\mathit{size}(\texttt{pred}\ \mathrm{t}) &= \mathit{size}(\mathrm{t}) + 1 \\
\mathit{size}(\texttt{iszero}\ \mathrm{t}) &= \mathit{size}(\mathrm{t}) + 1
\end{aligned}
$$

size

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

$$
\begin{aligned}
\mathit{size}(\mathtt{true}) &= 1 \\
\mathit{size}(\mathtt{false}) &= 1 \\
\mathit{size}(0) &= 1 \\
\mathit{size}(\mathtt{succ}\ \mathrm{t}) &= \mathit{size}(\mathrm{t}) + 1 \\
\mathit{size}(\mathtt{pred}\ \mathrm{t}) &= \mathit{size}(\mathrm{t}) + 1 \\
\mathit{size}(\mathtt{iszero}\ \mathrm{t}) &= \mathit{size}(\mathrm{t}) + 1 \\
\mathit{size}(\mathtt{if}\ \mathrm{t}_1\ \mathtt{then}\ \mathrm{t}_2\ \mathtt{else}\ \mathrm{t}_3) &= \mathit{size}(\mathrm{t}_1) + \mathit{size}(\mathrm{t}_2) + \mathit{size}(\mathrm{t}_3)
\end{aligned}
$$

depth

- The maximum depth of the abstract syntax tree of a term $\mathtt{t}$.
- Equivalently, the smallest *i* such that $\mathrm{t} \in \mathcal{S}_i$.

depth

- The maximum depth of the abstract syntax tree of a term $\mathtt{t}$.
- Equivalently, the smallest i such that $\mathtt{t} \in \mathcal{S}_i$.

$$\mathit{depth}(\mathtt{true}) \quad = \quad 1$$

depth

- The maximum depth of the abstract syntax tree of a term t.
- Equivalently, the smallest i such that $\mathrm{t} \in \mathcal{S}_i$.

$$
\begin{aligned}
\textit{depth}(\texttt{true}) &= 1 \\
\textit{depth}(\texttt{false}) &= 1
\end{aligned}
$$

depth

- The maximum depth of the abstract syntax tree of a term $\mathtt{t}$.
- Equivalently, the smallest i such that $\mathtt{t} \in \mathcal{S}_i$.

$$
\begin{aligned}
\mathit{depth}(\mathtt{true}) &= 1 \\
\mathit{depth}(\mathtt{false}) &= 1 \\
\mathit{depth}(0) &= 1
\end{aligned}
$$

depth

- The maximum depth of the abstract syntax tree of a term t.
- Equivalently, the smallest *i* such that $\mathrm{t} \in \mathcal{S}_i$.

$$
\begin{aligned}
\mathit{depth}(\mathtt{true}) &= 1 \\
\mathit{depth}(\mathtt{false}) &= 1 \\
\mathit{depth}(0) &= 1 \\
\mathit{depth}(\mathtt{succ}\ \mathrm{t}) &= \mathit{depth}(\mathrm{t}) + 1
\end{aligned}
$$

depth

- The maximum depth of the abstract syntax tree of a term t.
- Equivalently, the smallest i such that $\mathrm{t} \in \mathcal{S}_i$.

$$
\begin{aligned}
\mathit{depth}(\texttt{true}) &= 1 \\
\mathit{depth}(\texttt{false}) &= 1 \\
\mathit{depth}(0) &= 1 \\
\mathit{depth}(\texttt{succ}\ \mathrm{t}) &= \mathit{depth}(\mathrm{t}) + 1 \\
\mathit{depth}(\texttt{pred}\ \mathrm{t}) &= \mathit{depth}(\mathrm{t}) + 1
\end{aligned}
$$

depth

- The maximum depth of the abstract syntax tree of a term t.
- Equivalently, the smallest i such that $\mathrm{t} \in \mathcal{S}_i$.

$$
\begin{aligned}
depth(\texttt{true}) &= 1 \\
depth(\texttt{false}) &= 1 \\
depth(0) &= 1 \\
depth(\texttt{succ}\ \mathrm{t}) &= depth(\mathrm{t}) + 1 \\
depth(\texttt{pred}\ \mathrm{t}) &= depth(\mathrm{t}) + 1 \\
depth(\texttt{iszero}\ \mathrm{t}) &= depth(\mathrm{t}) + 1
\end{aligned}
$$

depth

- The maximum depth of the abstract syntax tree of a term t.
- Equivalently, the smallest i such that $\mathrm{t} \in \mathcal{S}_i$.

$$
\begin{aligned}
\mathit{depth}(\texttt{true}) &= 1 \\
\mathit{depth}(\texttt{false}) &= 1 \\
\mathit{depth}(0) &= 1 \\
\mathit{depth}(\texttt{succ}\ \mathrm{t}) &= \mathit{depth}(\mathrm{t}) + 1 \\
\mathit{depth}(\texttt{pred}\ \mathrm{t}) &= \mathit{depth}(\mathrm{t}) + 1 \\
\mathit{depth}(\texttt{iszero}\ \mathrm{t}) &= \mathit{depth}(\mathrm{t}) + 1 \\
\mathit{depth}(\texttt{if}\ \mathrm{t}_1\ \texttt{then}\ \mathrm{t}_2\ \texttt{else}\ \mathrm{t}_3) &= \max(\mathit{depth}(\mathrm{t}_1), \mathit{depth}(\mathrm{t}_2), \\
&\qquad \mathit{depth}(\mathrm{t}_3)) + 1
\end{aligned}
$$

A Simple Property of Terms

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

$$|\mathit{Consts}(\mathrm{t})| \leq \mathit{size}(\mathrm{t})$$

A Simple Property of Terms

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

$$|\mathit{Consts}(\mathrm{t})| \leq \mathit{size}(\mathrm{t})$$

- **Proof:** Exercise.

The Set of Values

$\mathbb{V} :=$ – *values*

The Set of Values

$\mathbf{v}$:=	– *values*
`true`	– *value true*

The Set of Values

v :=	– *values*
`true`	– *value true*
`false`	– *value false*

The Set of Values

V :=	– *values*
`true`	– *value true*
`false`	– *value false*
0	– *value zero*

The Set of Values

$\mathbf{v}$:=	– *values*
`true`	– *value true*
`false`	– *value false*
0	– *value zero*
`succ` $\mathbf{v}$	– *successor value*

Small-step Operational Semantics

- $\mathrm{t} \rightarrow \mathrm{t}'$ denotes "t evaluates to t' in one step"

Small-step Operational Semantics

- $\mathrm{t} \to \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\texttt{if true then } \mathrm{t}_2 \texttt{ else } \mathrm{t}_3 \to \mathrm{t}_2$$

Small-step Operational Semantics

- $\mathrm{t} \rightarrow \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\texttt{if true then } t_2 \texttt{ else } t_3 \rightarrow t_2$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \rightarrow t_3$$

Small-step Operational Semantics

- $\mathrm{t} \rightarrow \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\texttt{if true then } \mathrm{t}_2 \texttt{ else } \mathrm{t}_3 \rightarrow \mathrm{t}_2$$

$$\texttt{if false then } \mathrm{t}_2 \texttt{ else } \mathrm{t}_3 \rightarrow \mathrm{t}_3$$

$$\frac{\mathrm{t}_1 \rightarrow \mathrm{t}_1'}{\texttt{if } \mathrm{t}_1 \texttt{ then } \mathrm{t}_2 \texttt{ else } \mathrm{t}_3 \rightarrow \texttt{if } \mathrm{t}_1' \texttt{ then } \mathrm{t}_2 \texttt{ else } \mathrm{t}_3}$$

Small-step Operational Semantics (contd...)

- $\mathrm{t} \to \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\frac{\mathrm{t}_1 \to \mathrm{t}_1'}{\texttt{succ}\ \mathrm{t}_1 \to \texttt{succ}\ \mathrm{t}_1'}$$

Small-step Operational Semantics (contd...)

- $\mathrm{t} \to \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\frac{\mathrm{t}_1 \to \mathrm{t}_1'}{\texttt{succ}\ \mathrm{t}_1 \to \texttt{succ}\ \mathrm{t}_1'}$$

$$\texttt{pred}\ 0 \to 0$$

Small-step Operational Semantics (contd...)

- $\mathrm{t} \to \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\frac{\mathrm{t}_1 \to \mathrm{t}_1'}{\texttt{succ}\ \mathrm{t}_1 \to \texttt{succ}\ \mathrm{t}_1'}$$

$$\texttt{pred}\ 0 \to 0$$

$$\texttt{pred}\ (\texttt{succ}\ \mathrm{v}) \to \mathrm{v}$$

Small-step Operational Semantics (contd. . .)

- $t \to t'$ denotes "t evaluates to t' in one step"

$$\frac{t_1 \to t_1'}{\texttt{succ}\ t_1 \to \texttt{succ}\ t_1'}$$

$$\texttt{pred}\ 0 \to 0$$

$$\texttt{pred}\ (\texttt{succ}\ v) \to v$$

$$\frac{t_1 \to t_1'}{\texttt{pred}\ t_1 \to \texttt{pred}\ t_1'}$$

Small-step Operational Semantics (contd. . .)

- $\mathrm{t} \to \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\texttt{iszero}\ 0 \to \texttt{true}$$

Small-step Operational Semantics (contd. . .)

- $\mathrm{t} \to \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\texttt{iszero}\ 0 \to \texttt{true}$$

$$\texttt{iszero}\ (\texttt{succ}\ \mathrm{v}) \to \texttt{false}$$

Small-step Operational Semantics (contd...)

- $\mathrm{t} \rightarrow \mathrm{t}'$ denotes "t evaluates to t' in one step"

$$\texttt{iszero}\ 0 \rightarrow \texttt{true}$$

$$\texttt{iszero}\ (\texttt{succ}\ \mathrm{v}) \rightarrow \texttt{false}$$

$$\frac{\mathrm{t}_1 \rightarrow \mathrm{t}_1'}{\texttt{iszero}\ \mathrm{t}_1 \rightarrow \texttt{iszero}\ \mathrm{t}_1'}$$

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 $t \to t'$.

Evaluation Sequence

- An evaluation sequence starting from a term t is a (finite or infinite) sequence of terms $\mathrm{t}_1, \mathrm{t}_2, \ldots$, such that

$$\mathrm{t} \rightarrow \mathrm{t}_1$$

$$\mathrm{t}_1 \rightarrow \mathrm{t}_2$$

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”