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

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

t :=   – *terms*

    `true`   – *constant true*

    `false`   – *constant false*

    `if` t `then` t `else` t   – *conditional*

    0   – *constant zero*

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
```

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*

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

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:

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

$$\frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{succ}\ \texttt{t}_1 \in \mathcal{T}} \qquad \frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{pred}\ \texttt{t}_1 \in \mathcal{T}} \qquad \frac{\texttt{t}_1 \in \mathcal{T}}{\texttt{iszero}\ \texttt{t}_1 \in \mathcal{T}}$$

Syntax: Inference Rules

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

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

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

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

Concrete Syntax

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

Concrete Syntax

$$\begin{aligned}
\mathcal{S}_0 &= \emptyset \\
\mathcal{S}_{i+1} &= \quad \{\texttt{true}, \texttt{false}, \mathbf{0}\}
\end{aligned}$$

Concrete Syntax

$$\begin{aligned}
\mathcal{S}_0 &= \emptyset \\
\mathcal{S}_{i+1} &= \quad \{\texttt{true}, \texttt{false}, \texttt{0}\} \\
&\quad \cup \{\texttt{succ } t_1, \texttt{pred } t_1, \texttt{iszero } t_1 \mid t_1 \in \mathcal{S}_i\}
\end{aligned}$$

Concrete Syntax

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

Concrete Syntax

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

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

Concrete Syntax

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

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. $\in \{\texttt{true}, \texttt{false}, 0\}$

Induction on Terms

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

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.

Induction on Terms

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

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.

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

Consts

► The set of constants in a term t.

$$Consts(\texttt{true}) = \{\texttt{true}\}$$
$$Consts(\texttt{false}) = \{\texttt{false}\}$$

Consts

▶ The set of constants in a term t.

$$Consts(\texttt{true}) = \{\texttt{true}\}$$
$$Consts(\texttt{false}) = \{\texttt{false}\}$$
$$Consts(0) = \{0\}$$

Consts

▶ The set of constants in a term t.

$$Consts(\texttt{true}) = \{\texttt{true}\}$$
$$Consts(\texttt{false}) = \{\texttt{false}\}$$
$$Consts(0) = \{0\}$$
$$Consts(\texttt{succ t}) = Consts(\texttt{t})$$

Consts

- The set of constants in a term t.

$$Consts(\texttt{true}) = \{\texttt{true}\}$$
$$Consts(\texttt{false}) = \{\texttt{false}\}$$
$$Consts(0) = \{0\}$$
$$Consts(\texttt{succ } t) = Consts(t)$$
$$Consts(\texttt{pred } t) = Consts(t)$$

Consts

▶ The set of constants in a term t.

$$Consts(\texttt{true}) = \{\texttt{true}\}$$
$$Consts(\texttt{false}) = \{\texttt{false}\}$$
$$Consts(0) = \{0\}$$
$$Consts(\texttt{succ t}) = Consts(\texttt{t})$$
$$Consts(\texttt{pred t}) = Consts(\texttt{t})$$
$$Consts(\texttt{iszero t}) = Consts(\texttt{t})$$

Consts

- The set of constants in a term t.

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

- ► 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.

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

size

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

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

size

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

$$size(\texttt{true}) = 1$$
$$size(\texttt{false}) = 1$$
$$size(0) = 1$$

size

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

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

size

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

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

size

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

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

size

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

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

depth

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

depth

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

$$depth(\texttt{true}) \;=\; 1$$

depth

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

$$depth(\texttt{true}) = 1$$
$$depth(\texttt{false}) = 1$$

depth

▶ The maximum depth of the abstract syntax tree of a term t.

▶ Equivalently, the smallest i such that $t \in \mathcal{S}_i$.

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

depth

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

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

depth

▶ The maximum depth of the abstract syntax tree of a term t.

▶ Equivalently, the smallest i such that $t \in \mathcal{S}_i$.

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

depth

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

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

depth

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

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

A Simple Property of Terms

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

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

A Simple Property of Terms

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

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

▶ **Proof:** Exercise.

The Set of Values

V :=   – *values*

The Set of Values

$$V :=$$ – *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

V :=		– *values*
	true	– *value true*
	false	– *value false*
	0	– *value zero*
	succ **V**	– *successor value*

Small-step Operational Semantics

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

Small-step Operational Semantics

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

$$\text{if true then } t_2 \text{ else } t_3 \to t_2$$

Small-step Operational Semantics

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

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

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

Small-step Operational Semantics

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

$$\text{if true then } t_2 \text{ else } t_3 \to t_2$$

$$\text{if false then } t_2 \text{ else } t_3 \to t_3$$

$$\frac{t_1 \to t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \to \text{if } t_1' \text{ then } t_2 \text{ else } t_3}$$

Small-step Operational Semantics (contd...)

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

$$\frac{t_1 \rightarrow t_1'}{\text{succ } t_1 \ \rightarrow \text{succ } t_1'}$$

Small-step Operational Semantics (contd. . .)

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

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

$$\text{pred } 0 \to 0$$

Small-step Operational Semantics (contd. . .)

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

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

$$\mathrm{pred}\ 0 \to 0$$

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

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"

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

Small-step Operational Semantics (contd. . .)

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

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

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

Small-step Operational Semantics (contd. . .)

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

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

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

$$\frac{t_1 \rightarrow t_1'}{\texttt{iszero}\ t_1 \rightarrow \texttt{iszero}\ t_1'}$$

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 $t_1, t_2, \ldots$, 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"