barkhauseninstitut.org

Foundations of Certified Programming Language and Compiler Design

Dr.-Ing. Sebastian Ertel

Composable Operating Systems Group, Barkhausen Institute

Outline

Lecture	Logic	Formalisms	PL
1	Propositional and first-order logic		
2			Functional programming
3		Syntax and Semantics	
4			The untyped lambda calculus
5		Types	
6			The typed lambda calculus
7			Polymorphism
8		Curry-Howard	
9			Higher-order types
10			Dependent types

Main Goals

Remember evaluation may get stuck, i.e., fail at runtime! (`succ true`)

Desirable goal The compiler *verifies* statically (,i.e, at compile-time,) that my program does not get stuck.

Main Goals

Remember evaluation may get stuck, i.e., fail at runtime! (`succ true`)

Desirable goal The compiler *verifies* statically (,i.e, at compile-time,) that my program does not get stuck.

Let's enter the world of types and

- learn how to define a type system based upon which
- we can prove that evaluation of typed terms cannot get stuck.

Syntax of Arithmetic Expression with Booleans

Syntax

t	$::=$		terms:
	$\vert$	`true`	constant true
	$\vert$	`false`	constant false
	$\vert$	`if` t `then` t `else` t	conditional

v	$::=$		values:
	$\vert$	`true`	true value
	$\vert$	`false`	false value

Syntax of Arithmetic Expression with Booleans

Syntax

$$
\begin{array}{llll}
t & ::= & & \text{terms:} \\
 & | & \texttt{true} & \text{constant true} \\
 & | & \texttt{false} & \text{constant false} \\
 & | & \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t & \text{conditional}
\end{array}
\qquad
\begin{array}{llll}
v & ::= & & \text{values:} \\
 & | & \texttt{true} & \text{true value} \\
 & | & \texttt{false} & \text{false value}
\end{array}
$$

<u>Additional</u> Syntax

$$
\begin{array}{llll}
t & ::= & \ldots & \text{terms:} \\
 & | & 0 & \text{constant zero} \\
 & | & \texttt{succ}\ t & \text{successor} \\
 & | & \texttt{pred}\ t & \text{predecessor} \\
 & | & \texttt{iszero}\ t & \text{check}
\end{array}
\qquad
\begin{array}{llll}
v & ::= & \ldots & \text{values:} \\
 & | & nv & \text{numeric value}
\end{array}
\qquad
\begin{array}{llll}
nv & ::= & & \text{numeric values} \\
 & | & 0 & \text{zero value} \\
 & | & \texttt{succ}\ nv & \text{successor value}
\end{array}
$$

Examples

Terms $\texttt{if iszero}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{pred}\ (\texttt{succ}\ 0), \texttt{true}, 0, \ldots$

Examples

Terms	$\texttt{if iszero}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{pred}\ (\texttt{succ}\ 0), \texttt{true}, 0, \ldots$
"Other terms"	$\texttt{if}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{succ true},$

Examples

Terms $\texttt{if iszero } 0 \texttt{ then succ } 0 \texttt{ else } 0, \texttt{pred } (\texttt{succ } 0), \texttt{true}, 0, \ldots$

"Other terms" $\texttt{if } 0 \texttt{ then succ } 0 \texttt{ else } 0, \texttt{succ true},$

Boolean values $\texttt{true}, \texttt{false}$

Examples

Terms	$\texttt{if iszero } 0 \texttt{ then succ } 0 \texttt{ else } 0, \texttt{pred } (\texttt{succ } 0), \texttt{true}, 0, \ldots$
“Other terms”	$\texttt{if } 0 \texttt{ then succ } 0 \texttt{ else } 0, \texttt{succ true},$
Boolean values	$\texttt{true}, \texttt{false}$
Numeric Values	$0, \texttt{succ } 0, \texttt{succ succ } 0, \ldots$

Semantics

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

Semantics

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \text{ E-IF}$$

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

Semantics

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

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \longrightarrow \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3}\ \text{E-IF}$$

$$\frac{}{\texttt{if true then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_2}\ \text{E-IFTRUE}$$

$$\frac{}{\texttt{if false then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_3}\ \text{E-IFFALSE}$$

Semantics

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

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \longrightarrow \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3}\ \textsc{E-If}$$

$$\frac{}{\texttt{if true then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_2}\ \textsc{E-IfTrue}$$

$$\frac{}{\texttt{if false then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_3}\ \textsc{E-IfFalse}$$

$$\frac{t \longrightarrow t'}{\texttt{succ}\ t \longrightarrow \texttt{succ}\ t'}\ \textsc{E-Succ}$$

$$\frac{t \longrightarrow t'}{\texttt{pred}\ t \longrightarrow \texttt{pred}\ t'}\ \textsc{E-Pred}$$

Semantics

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

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \text{ E-IF}$$

$$\frac{}{\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2} \text{ E-IFTRUE} \qquad \frac{}{\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3} \text{ E-IFFALSE}$$

$$\frac{t \longrightarrow t'}{\texttt{succ } t \longrightarrow \texttt{succ } t'} \text{ E-SUCC} \qquad \frac{t \longrightarrow t'}{\texttt{pred } t \longrightarrow \texttt{pred } t'} \text{ E-PRED}$$

$$\frac{}{\texttt{pred } (\texttt{succ } nv) \longrightarrow nv} \text{ E-PREDSUCC} \qquad \frac{}{\texttt{pred } 0 \longrightarrow 0} \text{ E-PREDZERO}$$

Semantics

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

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \longrightarrow \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3}\ \textsc{E-If}$$

$$\frac{}{\texttt{if true then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_2}\ \textsc{E-IfTrue} \qquad \frac{}{\texttt{if false then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_3}\ \textsc{E-IfFalse}$$

$$\frac{t \longrightarrow t'}{\texttt{succ}\ t \longrightarrow \texttt{succ}\ t'}\ \textsc{E-Succ} \qquad \frac{t \longrightarrow t'}{\texttt{pred}\ t \longrightarrow \texttt{pred}\ t'}\ \textsc{E-Pred}$$

$$\frac{}{\texttt{pred}\ (\texttt{succ}\ nv) \longrightarrow nv}\ \textsc{E-PredSucc} \qquad \frac{}{\texttt{pred}\ 0 \longrightarrow 0}\ \textsc{E-PredZero}$$

$$\frac{t \longrightarrow t'}{\texttt{iszero}\ t \longrightarrow \texttt{iszero}\ t'}\ \textsc{E-IsZero1} \qquad \frac{}{\texttt{iszero}\ 0 \longrightarrow \texttt{true}}\ \textsc{E-IsZero2}$$

Semantics

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

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 \longrightarrow \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3}\ \text{E-IF}$$

$$\frac{}{\texttt{if true then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_2}\ \text{E-IFTRUE}$$

$$\frac{}{\texttt{if false then}\ t_2\ \texttt{else}\ t_3 \longrightarrow t_3}\ \text{E-IFFALSE}$$

$$\frac{t \longrightarrow t'}{\texttt{succ}\ t \longrightarrow \texttt{succ}\ t'}\ \text{E-SUCC}$$

$$\frac{t \longrightarrow t'}{\texttt{pred}\ t \longrightarrow \texttt{pred}\ t'}\ \text{E-PRED}$$

$$\frac{}{\texttt{pred}\ (\texttt{succ}\ nv) \longrightarrow nv}\ \text{E-PREDSUCC}$$

$$\frac{}{\texttt{pred}\ 0 \longrightarrow 0}\ \text{E-PREDZERO}$$

$$\frac{t \longrightarrow t'}{\texttt{iszero}\ t \longrightarrow \texttt{iszero}\ t'}\ \text{E-ISZERO1}$$

$$\frac{}{\texttt{iszero}\ 0 \longrightarrow \texttt{true}}\ \text{E-ISZERO2}$$

$$\frac{}{\texttt{iszero}\ (\texttt{succ}\ nv) \longrightarrow \texttt{false}}\ \text{E-ISZERO3}$$

Examples

Terms in $\longrightarrow$ $\texttt{if iszero}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{pred}\ (\texttt{succ}\ 0), \cancel{\texttt{true}}, \cancel{0}, \ldots$

Stuck terms $\texttt{if}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{succ true},$

Boolean values $\texttt{true}, \texttt{false}$

Numeric Values $0, \texttt{succ}\ 0, \texttt{succ succ}\ 0, \ldots$

Types

- Let's introduce two types, `Nat` or `Bool`, to classify the terms of our example language.

Types

- Let's introduce two types, `Nat` or `Bool`, to classify the terms of our example language.
- To refer to types in general, we use metavariables S, T, U.

Types

- Let's introduce two types, `Nat` or `Bool`, to classify the terms of our example language.
- To refer to types in general, we use metavariables S, T, U.
- We say "a term t has type T" and mean that t evaluates to a value of type T.

The Typing Relation

New syntactic forms:

The Typing Relation

New syntactic forms:

T	$::=$		types:
	$\vert$	`Bool`	type of booleans

T	$::=$	$\ldots$	types:
	$\vert$	`Nat`	type of natural numbers

The Typing Relation

New syntactic forms:

T	$::=$		types:	T	$::=$ $\dots$		types:
	$\vert$	`Bool`	type of booleans		$\vert$	`Nat`	type of natural numbers

New Typing Rules:

The Typing Relation

New syntactic forms:

T ::=	types:	T ::= ...	types:
| `Bool`	type of booleans	| `Nat`	type of natural numbers

New Typing Rules:

$\boxed{t : T}$

The Typing Relation

New syntactic forms:

T	$::=$		types:	T	$::=$	$\ldots$	types:
	$\mid$	$\mathtt{Bool}$	type of booleans		$\mid$	$\mathtt{Nat}$	type of natural numbers

New Typing Rules:

$\boxed{t : T}$

$$\frac{}{\mathtt{0} : \mathtt{Nat}} \quad \text{T-Zero} \qquad \frac{t : \mathtt{Nat}}{\mathtt{succ}\ t : \mathtt{Nat}} \quad \text{T-Succ} \qquad \frac{t : \mathtt{Nat}}{\mathtt{pred}\ t : \mathtt{Nat}} \quad \text{T-Pred}$$

The Typing Relation

New syntactic forms:

$$\begin{array}{llll} T & ::= & & \text{types:} \\ & | & \texttt{Bool} & \text{type of booleans} \end{array} \qquad \begin{array}{llll} T & ::= & \ldots & \text{types:} \\ & | & \texttt{Nat} & \text{type of natural numbers} \end{array}$$

New Typing Rules:

$\boxed{t : T}$

$$\frac{}{\texttt{0} : \texttt{Nat}}\ \text{T-Zero} \qquad \frac{t : \texttt{Nat}}{\texttt{succ}\ t : \texttt{Nat}}\ \text{T-Succ} \qquad \frac{t : \texttt{Nat}}{\texttt{pred}\ t : \texttt{Nat}}\ \text{T-Pred}$$

$$\frac{}{\texttt{true} : \texttt{Bool}}\ \text{T-True} \qquad \frac{}{\texttt{false} : \texttt{Bool}}\ \text{T-False} \qquad \frac{t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}\ \text{T-If}$$

The Typing Relation

New syntactic forms:

T	::=		types:
	$\mid$	`Bool`	type of booleans

T	::=	...	types:
	$\mid$	`Nat`	type of natural numbers

New Typing Rules:

$\boxed{t : T}$

$$\frac{}{0 : \texttt{Nat}} \ \text{T-Zero} \qquad \frac{t : \texttt{Nat}}{\texttt{succ}\ t : \texttt{Nat}} \ \text{T-Succ} \qquad \frac{t : \texttt{Nat}}{\texttt{pred}\ t : \texttt{Nat}} \ \text{T-Pred}$$

$$\frac{}{\texttt{true} : \texttt{Bool}} \ \text{T-True} \qquad \frac{}{\texttt{false} : \texttt{Bool}} \ \text{T-False} \qquad \frac{t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T} \ \text{T-If}$$

$$\frac{t : \texttt{Nat}}{\texttt{iszero}\ t : \texttt{Bool}} \ \text{T-IsZero}$$

Examples

Terms in $\longrightarrow$ $\texttt{if iszero}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{pred}\ (\texttt{succ}\ 0), \cancel{\texttt{true}}, \cancel{0}, \ldots$

Stuck terms $\texttt{if}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{succ true},$

Boolean values $\texttt{true}, \texttt{false}$

Numeric Values $0, \texttt{succ}\ 0, \texttt{succ succ}\ 0, \ldots$

Typed terms $\texttt{if iszero}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0 : \texttt{Nat}, \texttt{pred}\ (\texttt{succ}\ 0) : \texttt{Nat}, \texttt{true} : \texttt{Bool}, 0 : \texttt{Nat} \ldots$

The Typing Relation

Definition (Typing Relation)

Formally, the *typing relation* (for our example language) is the smallest binary relation between terms and types satisfying all instances of the associated rules.

The Typing Relation

Definition (Typing Relation)

Formally, the *typing relation* (for our example language) is the smallest binary relation between terms and types satisfying all instances of the associated rules.

Definition (Well-typedness)

A term t is *typable* (or *well-typed*) if there is some T such that $t : T$.

- Recursive definition to calculate the type of a term for each syntactic form:

Lemma (Inversion of the typing relation)

1. *If* $0 : R$ *then* $R = \mathtt{Nat}$.

- Recursive definition to calculate the type of a term for each syntactic form:

Lemma (Inversion of the typing relation)

1. *If* $0 : R$ *then* $R = \mathtt{Nat}$.
2. *If* $\mathtt{succ}\, t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.

- Recursive definition to calculate the type of a term for each syntactic form:

Lemma (Inversion of the typing relation)

1. *If* $0 : R$ *then* $R = \mathtt{Nat}$.
2. *If* $\mathtt{succ}\, t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.
3. *If* $\mathtt{pred}\, t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.

- Recursive definition to calculate the type of a term for each syntactic form:

Lemma (Inversion of the typing relation)

1. *If* $0 : R$ *then* $R = \mathtt{Nat}$.
2. *If* $\mathtt{succ}\, t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.
3. *If* $\mathtt{pred}\, t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.
4. *If* $\mathtt{true} : R$ *then* $R = \mathtt{Bool}$.

- Recursive definition to calculate the type of a term for each syntactic form:

Lemma (Inversion of the typing relation)

1. *If* $0 : R$ *then* $R = \mathtt{Nat}$.
2. *If* $\mathtt{succ}\,t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.
3. *If* $\mathtt{pred}\,t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.
4. *If* $\mathtt{true} : R$ *then* $R = \mathtt{Bool}$.
5. *If* $\mathtt{false} : R$ *then* $R = \mathtt{Bool}$.

- Recursive definition to calculate the type of a term for each syntactic form:

Lemma (Inversion of the typing relation)

1. If $0 : R$ then $R = \mathtt{Nat}$.
2. If $\mathtt{succ}\, t : R$ then $t : \mathtt{Nat}$ and $R = \mathtt{Nat}$.
3. If $\mathtt{pred}\, t : R$ then $t : \mathtt{Nat}$ and $R = \mathtt{Nat}$.
4. If $\mathtt{true} : R$ then $R = \mathtt{Bool}$.
5. If $\mathtt{false} : R$ then $R = \mathtt{Bool}$.
6. If $\mathtt{if}\, t_1\ \mathtt{then}\, t_2\ \mathtt{else}\, t_3 : R$ then $t_1 : \mathtt{Bool}, t_2 : R$ and $t_3 : R$.

Calculating Types

Lemma

- Recursive definition to calculate the type of a term for each syntactic form:

Lemma (Inversion of the typing relation)

1. *If* $0 : R$ *then* $R = \mathtt{Nat}$.
2. *If* $\mathtt{succ}\, t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.
3. *If* $\mathtt{pred}\, t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Nat}$.
4. *If* $\mathtt{true} : R$ *then* $R = \mathtt{Bool}$.
5. *If* $\mathtt{false} : R$ *then* $R = \mathtt{Bool}$.
6. *If* $\mathtt{if}\, t_1\ \mathtt{then}\, t_2\ \mathtt{else}\, t_3 : R$ *then* $t_1 : \mathtt{Bool}, t_2 : R$ *and* $t_3 : R$.
7. *If* $\mathtt{iszero}\, t : R$ *then* $t : \mathtt{Nat}$ *and* $R = \mathtt{Bool}$

Proof.

Immediate from the typing relation. □

Typing Derivations

Typing Derivations

$$\dfrac{\dfrac{}{\texttt{true} : \texttt{Bool}}\ \textsc{T-True} \qquad \dfrac{}{0 : \texttt{Nat}}\ \textsc{T-Zero} \qquad \dfrac{\dfrac{}{0 : \texttt{Nat}}\ \textsc{T-Zero}}{\texttt{succ}\ 0 : \texttt{Nat}}\ \textsc{T-Succ}}{\texttt{if true then}\ 0\ \texttt{else succ}\ 0 : \texttt{Nat}}\ \textsc{T-If}$$

Typing statements are formal assertions about the typing of the program.

Typing Derivations

$$\dfrac{\dfrac{}{\texttt{true} : \texttt{Bool}}\ \textsc{T-True} \qquad \dfrac{}{0 : \texttt{Nat}}\ \textsc{T-Zero} \qquad \dfrac{\dfrac{}{0 : \texttt{Nat}}\ \textsc{T-Zero}}{\texttt{succ}\ 0 : \texttt{Nat}}\ \textsc{T-Succ}}{\texttt{if true then}\ 0\ \texttt{else succ}\ 0 : \texttt{Nat}}\ \textsc{T-If}$$

Typing statements are formal assertions about the typing of the program.
Typing rules are implications between statements.

Typing Derivations

$$\dfrac{\dfrac{}{\texttt{true} : \texttt{Bool}}\ \textsc{T-True} \qquad \dfrac{}{0 : \texttt{Nat}}\ \textsc{T-Zero} \qquad \dfrac{\dfrac{}{0 : \texttt{Nat}}\ \textsc{T-Zero}}{\texttt{succ}\ 0 : \texttt{Nat}}\ \textsc{T-Succ}}{\texttt{if true then}\ 0\ \texttt{else succ}\ 0 : \texttt{Nat}}\ \textsc{T-If}$$

Typing statements are formal assertions about the typing of the program.
Typing rules are implications between statements.
Typing derivations are deducations based on typing rules.

Theorem (Uniqueness of Types)

Each term t has at most one type. That is, if t is typable, then its type is unique and there is only one derivation of this typing from the inference rules.

Typing Derivations

$$
\dfrac{\dfrac{}{\texttt{true} : \texttt{Bool}}\ \textsc{T-True} \qquad \dfrac{}{0 : \texttt{Nat}}\ \textsc{T-Zero} \qquad \dfrac{\dfrac{}{0 : \texttt{Nat}}\ \textsc{T-Zero}}{\texttt{succ}\ 0 : \texttt{Nat}}\ \textsc{T-Succ}}{\texttt{if true then}\ 0\ \texttt{else succ}\ 0 : \texttt{Nat}}\ \textsc{T-If}
$$

Typing statements are formal assertions about the typing of the program.
Typing rules are implications between statements.
Typing derivations are deducations based on typing rules.

Theorem (Uniqueness of Types)

Each term t has at most one type. That is, if t is typable, then its type is unique and there is only one derivation of this typing from the inference rules.

Proof.

Structural induction on t with applications of the inversion lemma and the induction hypotheses. $\square$

Safety

- Well-typed terms do not "go wrong".

[1] In most systems, the resulting term has the same type.

Safety

- Well-typed terms do not “go wrong”.
- *Safety* is also called *soundness*.

[1] In most systems, the resulting term has the same type.

Safety

- Well-typed terms do not “go wrong”.
- *Safety* is also called *soundness*.
- Safety $=$ Progress $+$ Preservation

[1] In most systems, the resulting term has the same type.

Safety

- Well-typed terms do not “go wrong”.
- *Safety* is also called *soundness*.
- Safety $=$ Progress $+$ Preservation

[1] In most systems, the resulting term has the same type.

Safety

- Well-typed terms do not “go wrong”.
- *Safety* is also called *soundness*.
- Safety $=$ Progress $+$ Preservation

Definition (Progress)

A well-typed term is not stuck (either it is a value or it can take a step according to the evaluation rules.)

[1] In most systems, the resulting term has the same type.

Safety

- Well-typed terms do not “go wrong”.
- *Safety* is also called *soundness*.
- Safety $=$ Progress $+$ Preservation

Definition (Progress)

A well-typed term is not stuck (either it is a value or it can take a step according to the evaluation rules.)

[1] In most systems, the resulting term has the same type.

Safety

- Well-typed terms do not “go wrong”.
- *Safety* is also called *soundness*.
- Safety $=$ Progress $+$ Preservation

Definition (Progress)

A well-typed term is not stuck (either it is a value or it can take a step according to the evaluation rules.)

Definition (Preservation)

If a well-typed term takes a step of evaluation, then the resulting term is also well-typed.[1]

[1] In most systems, the resulting term has the same type.

Canonical Forms

Lemma

When proving progress, it is helpful to identify the well-typed terms that are values for the types in our language.

When proving progress, it is helpful to identify the well-typed terms that are values for the types in our language.

Lemma (Canonical Forms)

1. *If* $v : \texttt{Bool}$ *is a value (of type* $\texttt{Bool}$*) then* $v = \texttt{true} \lor v = \texttt{false}$.

When proving progress, it is helpful to identify the well-typed terms that are values for the types in our language.

Lemma (Canonical Forms)

1. *If* $v : \mathtt{Bool}$ *is a value (of type* $\mathtt{Bool}$*) then* $v = \mathtt{true} \lor v = \mathtt{false}$.
2. *If* $v : \mathtt{Nat}$ *is a value (of type* $\mathtt{Nat}$*) then* $v = 0 \lor v = \mathtt{succ}\, v_1$ *where* $v_1 : \mathtt{Nat}$ *is a value.*

Examples

Terms in $\longrightarrow$ $\texttt{if iszero}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{pred}\ (\texttt{succ}\ 0), \cancel{\texttt{true}}, \cancel{0}, \ldots$

Stuck terms $\texttt{if}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0, \texttt{succ true},$

Boolean values $\texttt{true}, \texttt{false}$

Numeric Values $0, \texttt{succ}\ 0, \texttt{succ succ}\ 0, \ldots$

Typed terms $\texttt{if iszero}\ 0\ \texttt{then succ}\ 0\ \texttt{else}\ 0 : \texttt{Nat}, \texttt{pred}\ (\texttt{succ}\ 0) : \texttt{Nat}, \texttt{true} : \texttt{Bool}, 0 : \texttt{Nat} \ldots$

Canonical forms $\texttt{true} : \texttt{Bool}, \texttt{false} : \texttt{Bool}, 0 : \texttt{Nat}, \texttt{succ}\ 0 : \texttt{Nat}, \texttt{succ succ}\ 0 : \texttt{Nat}, \ldots$

Canonical Forms
Proof

- Remember what we do: We connect terms that are defined as values to types.
- Mind the difference between the value definition $\texttt{succ}\ nv$ and the term definition $\texttt{succ}\ t$!

- Remember what we do: We connect terms that are defined as values to types.
- Mind the difference between the value definition $\texttt{succ}\ nv$ and the term definition $\texttt{succ}\ t$!

Proof.

- Values/(valued terms): $\texttt{true}$, $\texttt{false}$, 0, $\texttt{succ}\ n$ where n is a numeric value.

$\square$

Canonical Forms
Proof

- Remember what we do: We connect terms that are defined as values to types.
- Mind the difference between the value definition $\texttt{succ}\ nv$ and the term definition $\texttt{succ}\ t$!

Proof.

- Values/(valued terms): $\texttt{true}$, $\texttt{false}$, 0, $\texttt{succ}\ n$ where n is a numeric value.
- Applying the inversion lemma to connect values(/terms) to types, we get:

$\square$

Canonical Forms
Proof

- Remember what we do: We connect terms that are defined as values to types.
- Mind the difference between the value definition $\texttt{succ}\ nv$ and the term definition $\texttt{succ}\ t$!

Proof.

- Values/(valued terms): $\texttt{true}$, $\texttt{false}$, 0, $\texttt{succ}\ n$ where n is a numeric value.
- Applying the inversion lemma to connect values(/terms) to types, we get:
  1. the only boolean-typed terms that are values are: $\texttt{true} : \texttt{Bool}$, $\texttt{false} : \texttt{Bool}$

$\square$

Canonical Forms
Proof

- Remember what we do: We connect terms that are defined as values to types.
- Mind the difference between the value definition $\texttt{succ}\ nv$ and the term definition $\texttt{succ}\ t$!

Proof.

- Values/(valued terms): $\texttt{true}$, $\texttt{false}$, 0, $\texttt{succ}\ n$ where n is a numeric value.
- Applying the inversion lemma to connect values(/terms) to types, we get:
  1. the only boolean-typed terms that are values are: $\texttt{true} : \texttt{Bool}$, $\texttt{false} : \texttt{Bool}$
  2. the only numeric-typed terms that are values are: 0, $\texttt{succ}\ n$ where n is a value of type $\texttt{Nat}$

$\square$

Theorem (Progress)

If $t : T$ then either t is a value or there exists a t' such that $t \longrightarrow t'$.

Progress
Proof

- The proof is by induction on the typing derivation for $t : T$.

Progress
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-TRUE	$\dfrac{}{\texttt{true} : \texttt{Bool}}$ T-TRUE	Immediate.
T-FALSE	$\dfrac{}{\texttt{false} : \texttt{Bool}}$ T-FALSE	Immediate.

Progress

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-TRUE	$\dfrac{}{\texttt{true} : \texttt{Bool}}$ T-TRUE	Immediate.
T-FALSE	$\dfrac{}{\texttt{false} : \texttt{Bool}}$ T-FALSE	Immediate.
T-ZERO	$\dfrac{}{\texttt{0} : \texttt{Nat}}$ T-ZERO	Immediate.

Progress
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*

Progress
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-IF	By induction hypothesis:

Progress
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-IF	By induction hypothesis: **1)** If t_1 is a value then

Progress

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-If	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-If	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = \texttt{true}$ or $t_1 = \texttt{false}$.

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-IF	$$\frac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}\ \text{T-IF}$$	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = \texttt{true}$ or $t_1 = \texttt{false}$. By E-IFTRUE or E-IFFALSE we have $t \longrightarrow t'$ where $t' = t_2$ or $t' = t_3$.

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T}$ T-IF	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = \texttt{true}$ or $t_1 = \texttt{false}$. By E-IFTRUE or E-IFFALSE we have $t \longrightarrow t'$ where $t' = t_2$ or $t' = t_3$. **2)** If $t_1 \longrightarrow t_1'$ then

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-IF	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = \texttt{true}$ or $t_1 = \texttt{false}$. By E-IFTRUE or E-IFFALSE we have $t \longrightarrow t'$ where $t' = t_2$ or $t' = t_3$. **2)** If $t_1 \longrightarrow t_1'$ then by E-IF $t \longrightarrow t'$ where $t' = \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3$.

Progress

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-Succ	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-Succ	By induction hypothesis:

Progress

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-SUCC	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-SUCC	By induction hypothesis: **1)** If t_1 is a value then

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* $t{:}T$	*Proof*
T-Succ	$$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}\ \text{T-Succ}$$	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = 0$ or $t_1 = \texttt{succ}\ nv$.

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-Succ	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-Succ	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = 0$ or $t_1 = \texttt{succ}\ nv$. By the syntax definition of values (nv ::=) we have that $t = \texttt{succ}\ t_1$ is a value.

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* $t{:}T$	*Proof*
T-Succ	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-Succ	By induction hypothesis:
		1) If t_1 is a value then
		by the Canonical Forms lemma: $t_1 = 0$ or $t_1 = \texttt{succ}\ nv$.
		By the syntax definition of values ($nv\ ::=$) we have that $t = \texttt{succ}\ t_1$ is a value.
		2) If $t_1 \longrightarrow t_1'$ then

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-SUCC	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-SUCC	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = 0$ or $t_1 = \texttt{succ}\ nv$. By the syntax definition of values (nv ::=) we have that $t = \texttt{succ}\ t_1$ is a value. **2)** If $t_1 \longrightarrow t_1'$ then by E-SUCC we have $t \longrightarrow t'$ where $t' = \texttt{succ}\ t_1'$.

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*

Progress

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-PRED	$\dfrac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}}$ T-PRED	By induction hypothesis:

Progress
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-PRED	$\dfrac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}}$ T-PRED	By induction hypothesis: **1)** If t_1 is a value then

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-PRED	$\dfrac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}}$ T-PRED	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = 0$ or $t_1 = \texttt{succ}\ nv$ (because $t_1 : \texttt{Nat}$) such that

Progress
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-PRED	$\dfrac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}}$ T-PRED	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = 0$ or $t_1 = \texttt{succ}\ nv$ (because $t_1 : \texttt{Nat}$) such that either E-PREDZERO or E-PREDSUCC applies.

Progress

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-PRED	$\dfrac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}}$ T-PRED	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = 0$ or $t_1 = \texttt{succ}\ nv$ (because $t_1 : \texttt{Nat}$) such that either E-PREDZERO or E-PREDSUCC applies. **2)** If $t_1 \longrightarrow t_1'$ then

Progress
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-PRED	$\dfrac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}}$ T-PRED	By induction hypothesis: **1)** If t_1 is a value then by the Canonical Forms lemma: $t_1 = 0$ or $t_1 = \texttt{succ}\ nv$ (because $t_1 : \texttt{Nat}$) such that either E-PREDZERO or E-PREDSUCC applies. **2)** If $t_1 \longrightarrow t_1'$ then by E-PRED $t \longrightarrow \texttt{pred}\ t_1'$.

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-IsZero	$\dfrac{t_1 : \texttt{Nat}}{\texttt{iszero}\ t_1 : \texttt{Bool}}$ T-IsZero	By induction hypothesis: ... homework.

$\square$

Theorem (Preservation)

If $t : T$ *and* $t \longrightarrow t'$ *then* $t' : T$.

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-TRUE	$\dfrac{}{\texttt{true} : \texttt{Bool}}$ T-TRUE	t is a value such that there is no t' and the theorem holds.

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-TRUE	$\dfrac{}{\texttt{true} : \texttt{Bool}}$ T-TRUE	t is a value such that there is no t' and the theorem holds.
T-FALSE	$\dfrac{}{\texttt{false} : \texttt{Bool}}$ T-FALSE	t is a value such that there is no t' and the theorem holds.

Preservation

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-TRUE	$\dfrac{}{\texttt{true} : \texttt{Bool}}$ T-TRUE	t is a value such that there is no t' and the theorem holds.
T-FALSE	$\dfrac{}{\texttt{false} : \texttt{Bool}}$ T-FALSE	t is a value such that there is no t' and the theorem holds.
T-ZERO	$\dfrac{}{\texttt{0} : \texttt{Nat}}$ T-ZERO	t is a value such that there is no t' and the theorem holds.

Preservation

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* $t{:}T$	*Proof*

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-IF	We assume subderivations for t_1, t_2, t_3. 3 possible evaluation rules for t:

Preservation

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-IF	We assume subderivations for t_1, t_2, t_3. 3 possible evaluation rules for t: **1)** By E-IFTRUE we know that $t_1 = \texttt{true}$ such that $t \longrightarrow t_2$. By T-IF we have that $t_2 : T$.

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-IF	We assume subderivations for t_1, t_2, t_3. 3 possible evaluation rules for t: **1)** By E-IFTRUE we know that $t_1 = \texttt{true}$ such that $t \longrightarrow t_2$. By T-IF we have that $t_2 : T$. **2)** By E-IFFALSE we know that $t_1 = \texttt{false}$ such that $t \longrightarrow t_3$. By T-IF we have that $t_3 : T$.

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T}$ T-IF	We assume subderivations for t_1, t_2, t_3. 3 possible evaluation rules for t:
		1) By E-IFTRUE we know that $t_1 = \texttt{true}$ such that $t \longrightarrow t_2$. By T-IF we have that $t_2 : T$.
		2) By E-IFFALSE we know that $t_1 = \texttt{false}$ such that $t \longrightarrow t_3$. By T-IF we have that $t_3 : T$.
		3) By E-IF we know that $t_1 \longrightarrow t_1'$ such that $t \longrightarrow t'$ where $t' = \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3$.

Preservation

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-IF	We assume subderivations for t_1, t_2, t_3. 3 possible evaluation rules for t: **1)** By E-IFTRUE we know that $t_1 = \texttt{true}$ such that $t \longrightarrow t_2$. By T-IF we have that $t_2 : T$. **2)** By E-IFFALSE we know that $t_1 = \texttt{false}$ such that $t \longrightarrow t_3$. By T-IF we have that $t_3 : T$. **3)** By E-IF we know that $t_1 \longrightarrow t_1'$ such that $t \longrightarrow t'$ where $t' = \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3$. By T-IF we have that $t_2 : T, t_3 : T$.

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T}$ T-IF	We assume subderivations for t_1, t_2, t_3. 3 possible evaluation rules for t: **1)** By E-IFTRUE we know that $t_1 = \texttt{true}$ such that $t \longrightarrow t_2$. By T-IF we have that $t_2 : T$. **2)** By E-IFFALSE we know that $t_1 = \texttt{false}$ such that $t \longrightarrow t_3$. By T-IF we have that $t_3 : T$. **3)** By E-IF we know that $t_1 \longrightarrow t_1'$ such that $t \longrightarrow t'$ where $t' = \texttt{if}\ t_1'\ \texttt{then}\ t_2\ \texttt{else}\ t_3$. By T-IF we have that $t_2 : T, t_3 : T$.

Preservation

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-IF	$\dfrac{t_1 : \texttt{Bool} \quad t_2 : \texttt{T} \quad t_3 : \texttt{T}}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T}$ T-IF	We assume subderivations for t_1, t_2, t_3. 3 possible evaluation rules for t: **1)** By E-IFTRUE we know that $t_1 = \texttt{true}$ such that $t \longrightarrow t_2$. By T-IF we have that $t_2 : T$. **2)** By E-IFFALSE we know that $t_1 = \texttt{false}$ such that $t \longrightarrow t_3$. By T-IF we have that $t_3 : T$. **3)** By E-IF we know that $t_1 \longrightarrow t_1'$ such that $t \longrightarrow t'$ where $t' = \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3$. By T-IF we have that $t_2 : T, t_3 : T$.

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-SUCC	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-SUCC	We assume a subderivation of t_1. One possible rule exists:

Preservation

Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-Succ	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-Succ	We assume a subderivation of t_1. One possible rule exists: By E-Succ we know that $t \longrightarrow t'$ where $t' = \texttt{succ}\ t_1'$.

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with $t{:}T$*	*Proof*
T-Succ	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-Succ	We assume a subderivation of t_1. One possible rule exists: By E-Succ we know that $t \longrightarrow t'$ where $t' = \texttt{succ}\ t_1'$. By induction hypothesis, we have that $t_1 \longrightarrow t_1'$ with $t_1' : \texttt{Nat}$.

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with* t:T	*Proof*
T-Succ	$\dfrac{t_1 : \texttt{Nat}}{\texttt{succ}\ t_1 : \texttt{Nat}}$ T-Succ	We assume a subderivation of t_1. One possible rule exists: By E-Succ we know that $t \longrightarrow t'$ where $t' = \texttt{succ}\ t_1'$. By induction hypothesis, we have that $t_1 \longrightarrow t_1'$ with $t_1' : \texttt{Nat}$. By T-Succ on $\texttt{succ}\ t_1'$, we have that $t' : \texttt{Nat}$,i.e., $t' : T$.

Preservation
Proof

- The proof is by induction on the typing derivation for $t : T$.
- Cases:

Case	*Rule with t:T*	*Proof*
T-PRED	$\dfrac{t_1 : \texttt{Nat}}{\texttt{pred}\ t_1 : \texttt{Nat}}$ T-PRED	Homework.
T-ISZERO	$\dfrac{t_1 : \texttt{Nat}}{\texttt{iszero}\ t_1 : \texttt{Bool}}$ T-ISZERO	Homework.

$\square$

What we have learned

We have entered the world of types, i.e., compile-time verification:

- We know what a typing relation is: it prevents runtime errors, i.e., stuck evaluation.
- We know how to connect terms and types.
- We understand how to prove the first vital property of a typed language at compile-time: safety.