Types in Programming Languages (1)

Christian Urban

Wednesdays 10.15 - 11.45, Zuse

http://www4.in.tum.de/lehre/vorlesungen/types/WS0607/

Quotes

- Robin Milner in Computing Tomorrow:

  "One of the most helpful concepts in the whole of programming is the notion of type, used to classify the kinds of object which are manipulated. A significant proportion of programming mistakes are detected by an implementation which does type-checking before it runs any program."

- Leslie Lamport in Types Considered Harmful:

  "...mathematicians have gotten along quite well for two thousand years without types, and they still can today."

Learning Goals

At the end you

- can make up your own mind about types
- know about the issues with type-systems
- can define type-systems, implement type-checkers
- can prove properties about type-systems !

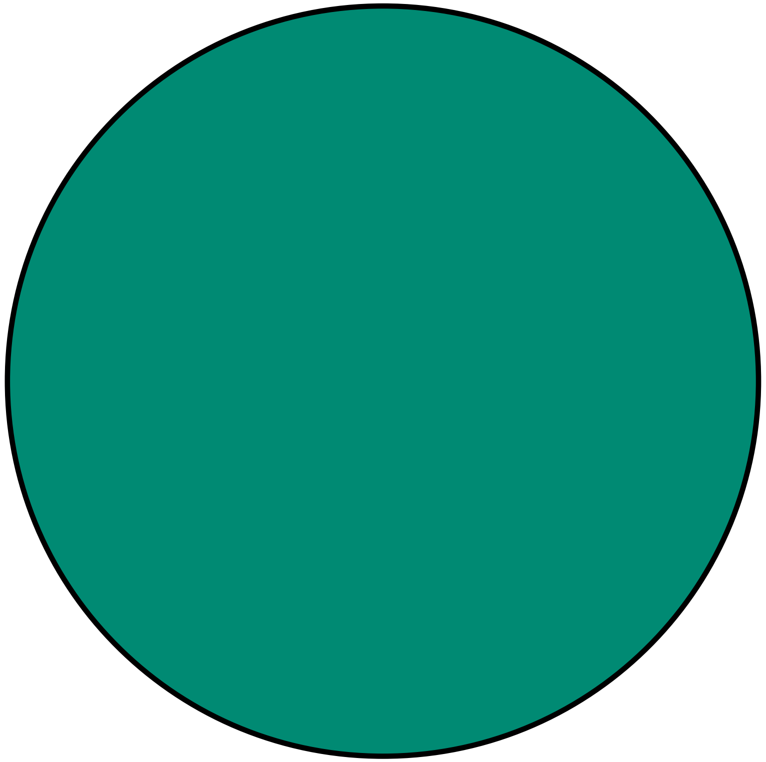
What Are Types Good For

- **Detect errors via type-checking** (prevent multiplication of an integer by a bool)
- **Abstraction and Interfaces** (programmer 1: “please give me a value in mph”; programmer 2: “I give you a value in kmph”)
- **Documentation** (useful hints about intended use which is kept consistent with the changes of the program)
- **Efficiency** (if I know a value is an int, I can compile to use machine registers)

Avoiding Embarrassing Claims

- C, C++, Java, Ocaml, SML, C#, F# all have types.
- Q: What is the difference between them?
- A: Some are better because they have a **strong** type-system. (In C you can use an integer as a bool via pointers. This defeats the purpose of types.)
- Q: But what about languages like LISP which have no types at all? Are they really really bad?

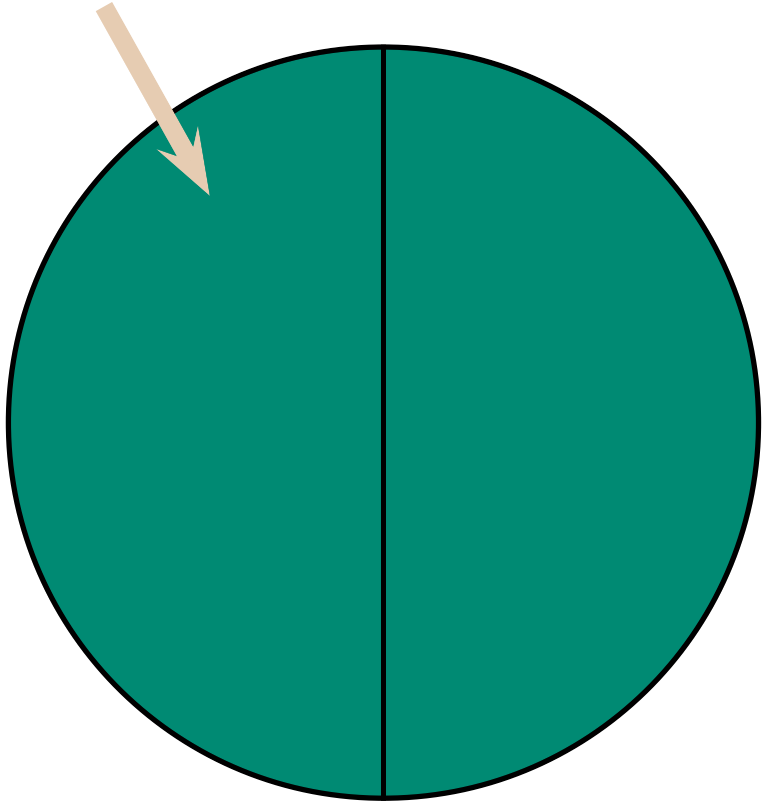
Errors



Errors

untrapped errors

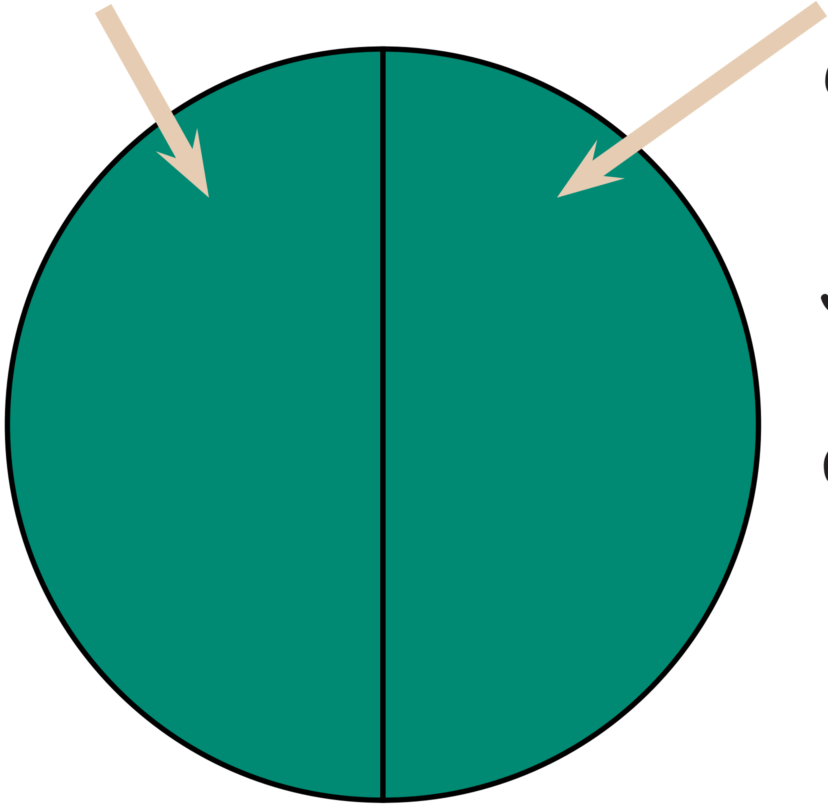
e.g. access of an array outside its bounds; jumping to a legal address



Errors

untrapped errors

e.g. access of an array outside its bounds; jumping to a legal address



trapped errors

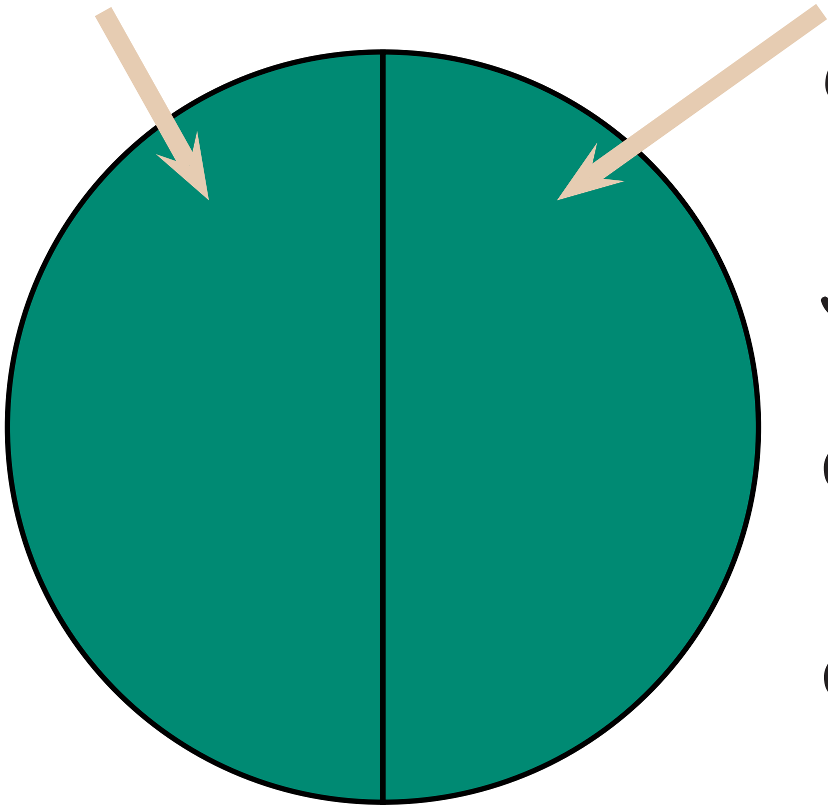
e.g. division by zero; jumping to an illegal address

Errors

untrapped errors

e.g. access of an array outside its bounds; jumping to a legal address

evil



trapped errors

e.g. division by zero; jumping to an illegal address

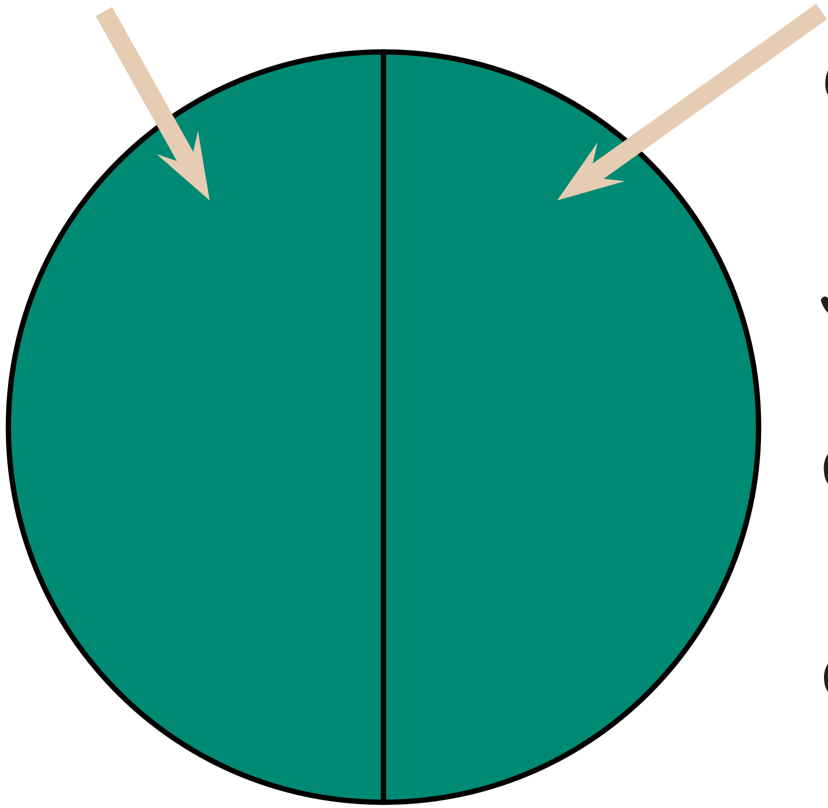
annoying

Errors

untrapped errors

e.g. access of an array outside its bounds; jumping to a legal address

evil



trapped errors

e.g. division by zero; jumping to an illegal address

annoying

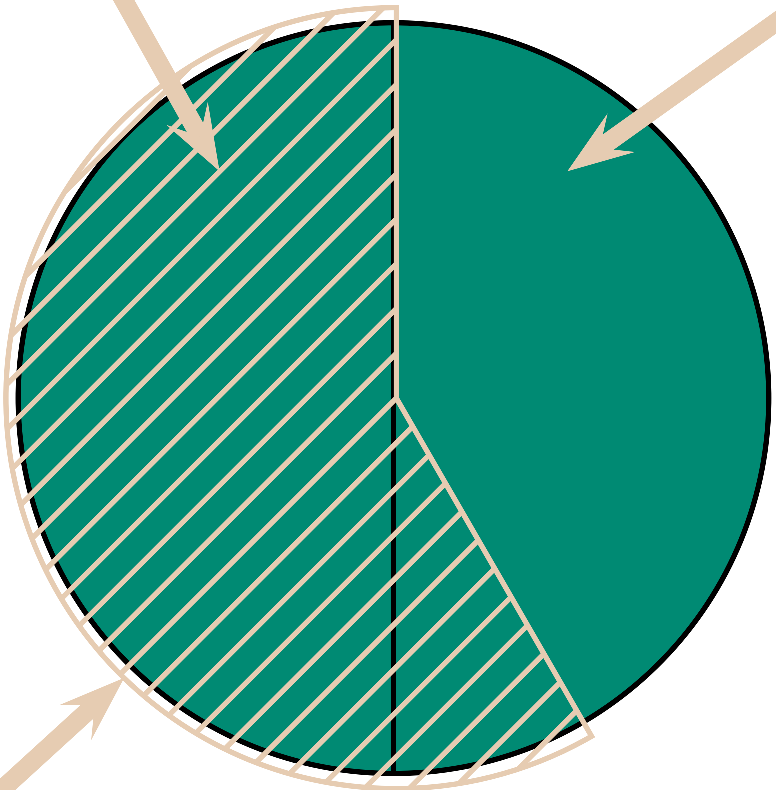
A programming language is called safe if no untrapped errors can occur. Safety can be achieved by run-time checks or static checks.

Errors

untrapped errors

e.g. access of an array outside its bounds; jumping to a legal address

evil



trapped errors

e.g. division by zero; jumping to an illegal address

annoying

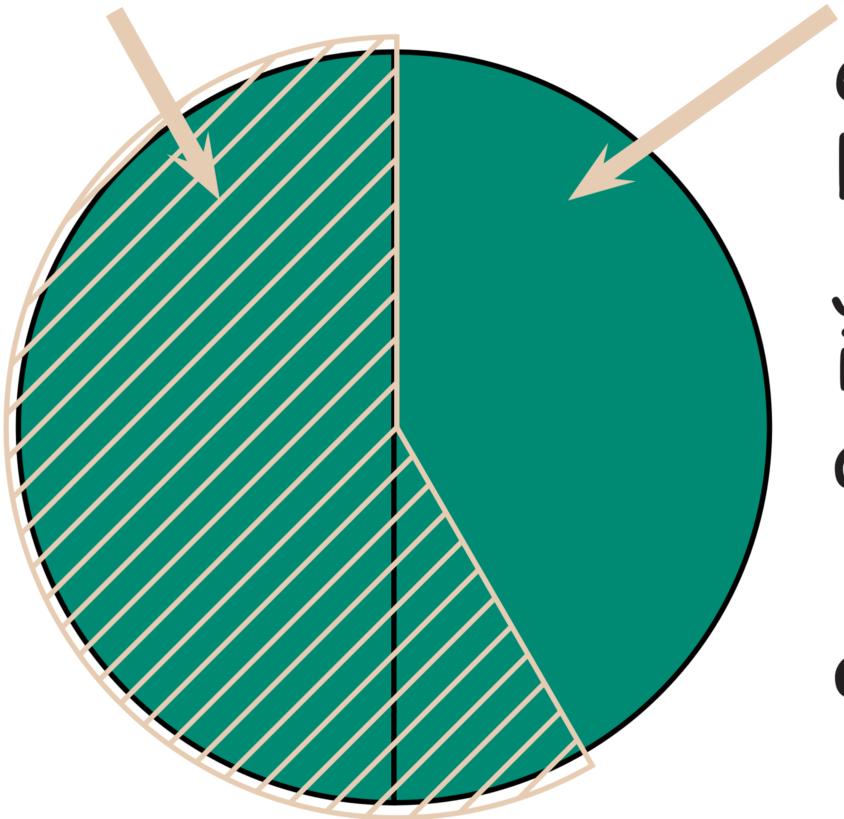
Forbidden errors include all untrapped errors and some trapped ones. A strongly typed programming language prevents all forbidden errors.

Errors

untrapped errors

e.g. access of an array outside its bounds; jumping to a legal address

evil



trapped errors

e.g. division by zero; jumping to an illegal address

annoying

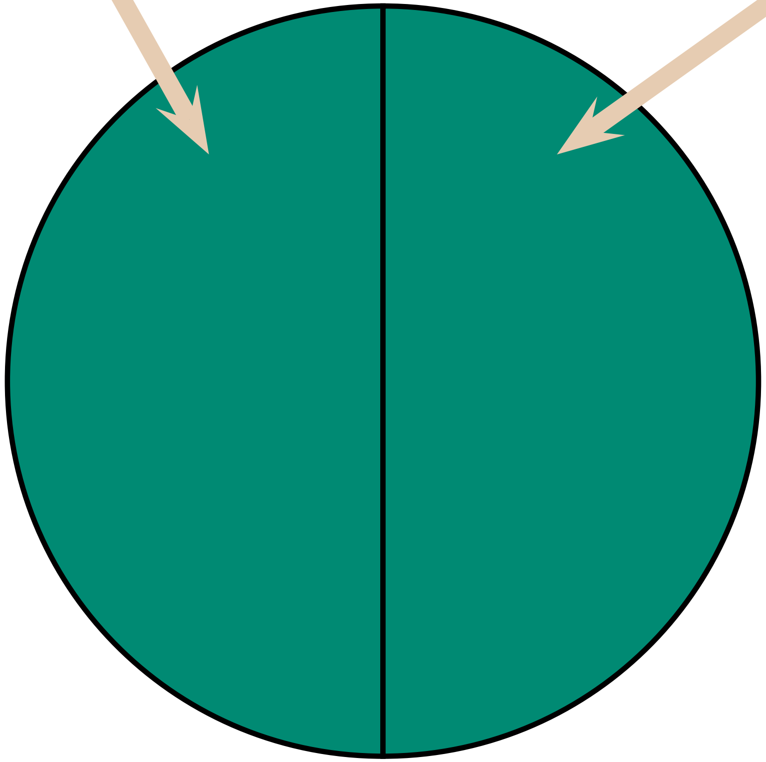
A weakly typed programming language prevents some untrapped errors, but **not** all; C, C++ have features that make them weakly typed.

Errors

untrapped errors

e.g. access of an array outside its bounds; jumping to a legal address

evil



trapped errors

e.g. division by zero; jumping to an illegal address

annoying

	Typed	Untyped
Safe	SML, Java	LISP
Unsafe	C, C++	Assembler

From "The Ten Commandments for C Programmers"

1) Thou shalt run **lint** [etc.] frequently and study its pronouncements with care, for verily its perception and judgement oft exceed thine.
2) Thou shalt not follow the NULL pointer, for chaos and madness await thee at its end.
3) Thou shalt cast all function arguments to the expected type if they are not of that type already, even when thou art convinced that this is unnecessary, lest they take cruel vengeance upon thee when thou least expect it.
4) If thy header files fail to declare the return types of thy library functions, thou shalt declare them thyself with the most meticulous care, lest grievous harm befall thy program.

From "The Ten Commandments for C Programmers"

1) Thou shalt run **lint** [etc.] frequently and study its pronouncements with care, for verily its perception and judgement oft exceed thine.

2) Thou shalt not follow the NULL pointer, for chaos and m

3) Thou s
expec
even w
unnecessary, lest they take cruel vengeance upon thee when thou least expect it.

Moral: Why not using a strongly typed programming language in the first place?

4) If thy header files fail to declare the return types of thy library functions, thou shalt declare them thyself with the most meticulous care, lest grievous harm befall thy program.

Expected Properties of Type Systems

Checks can be made statically or during run-time.
The checks should be:

- **decidable** (The purpose of types is not just stating the programmers intentions, but to prevent error.)
- **transparent** (Why a program type-checks or not should be predictable.)
- **should not be in the way in programming** (polymorphism)

Expected Properties of Type Systems

Checks can be made statically or during run-time.
The checks should be:

- **decidable** (The purpose of types is not just stating the programmers intentions, but to prevent error.)
- **transparent** (Why a program type-checks or not s
- **shoul**
  (poly

That means sometimes checks have to be done dynamically during run-time—programs become slower.

Expected Properties of Type Systems

Checks can be made statically or during run-time.
The checks should be:

- **decidable** (The purpose of types is not just stating the programmers intentions, but to prevent error.)
- **transparent** (Why a program type-checks or not should be predictable.)
- **should not be in the way in programming** (polymorphism)

Expected Properties of Type Systems

Checks can be made statically or during run-time.
The checks should be:

- **decidable** (The purpose of types is not just stating the programmers intentions, but to prevent error.)
- **transparent** (Why a program type-checks or not should be predictable.)
- **should n**
  (polymor

"This program contains a type-error" is not helpful for the programmer.

Expected Properties of Type Systems

Checks can be made statically or during run-time.
The checks should be:

- **decidable** (The purpose of types is not just stating the programmers intentions, but to prevent error.)
- **transparent** (Why a program type-checks or not should be predictable.)
- **should not be in the way in programming** (polymorphism)

Formal Specification of Type Systems

- **should provide a precise mathematical characterisation**
- **basis for type-soundness proofs** (It is quite difficult to design a strongly-typed language. We will see examples where people got it wrong.)
- **should keep algorithmic concerns and specification separate**

Example

To warm up, let's start with an example:

e	$::=$	x	variables
	$\mid$	true	
	$\mid$	false	
	$\mid$	gr e e	greater than
	$\mid$	le e e	less than
	$\mid$	eq e e	equal
	$\mid$	if e e e	if-then-else
	$\mid$	0	
	$\mid$	succ e	successor
	$\mid$	iszero e	

Example

To warm up, let's start with an example:

e	$::=$	x	variables
	$\mid$	true	
	$\mid$	false	
	$\mid$	gr e e	greater than
	$\mid$	le e e	less than
	$\mid$	eq e e	equal
	$\mid$	if e e e	if-then-else
	$\mid$	0	
	$\mid$	iszero e	

true, false, gr and so on are called constructors.

Possible Expressions

iszero (succ 0)

if true false true

if (iszero n) (succ 0) 0

gr 0 (succ 0)

iszero false

if 0 0 (succ 0)

if x 0 false

le true false

eq true (succ 0)

Possible Expressions

iszero (succ 0)

if true false true

if (iszero n) (succ 0) 0

gr 0 (succ 0)

iszero false

if 0 0 (succ 0)

if x 0 false

le true false

eq true (succ 0)

however these expressions look wrong

Typing

We introduce types bool and nat and a judgement:

- $$\frac{}{\mathsf{true} : \mathsf{bool}} \qquad \frac{}{\mathsf{false} : \mathsf{bool}} \qquad \frac{}{0 : \mathsf{nat}}$$

- iszero should only work over nats and produce a bool:

$$\frac{e : \mathsf{nat}}{\mathsf{iszero}\ e : \mathsf{bool}}$$

- $$\frac{e_1 : \mathsf{nat} \quad e_2 : \mathsf{nat}}{\mathsf{gr}\ e_1\ e_2 : \mathsf{bool}} \qquad \frac{e_1 : \mathsf{nat} \quad e_2 : \mathsf{nat}}{\mathsf{le}\ e_1\ e_2 : \mathsf{bool}}$$

$$\frac{e : \mathsf{nat}}{\mathsf{succ}\ e : \mathsf{nat}}$$

Typing

- T is a variable standing either for bool or for nat.

$$\frac{e_1 : \mathsf{bool} \quad e_2 : T \quad e_3 : T}{\mathsf{if}\ e_1\ e_2\ e_3 : T}$$

-

$$\frac{e_1 : T \quad e_2 : T}{\mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}$$

Type of Variables

What about variables?

$$\frac{}{x : T}$$

but

$$\frac{\frac{}{x : \mathsf{bool}} \qquad \frac{}{0 : \mathsf{nat}} \qquad \frac{\frac{}{x : \mathsf{nat}}}{\mathsf{succ}\ x : \mathsf{nat}}}{\mathsf{if}\ x\ 0\ (\mathsf{succ}\ x) : \mathsf{nat}}$$

Variables should refer to a single value (stored in a register or memory location)

Type-Contexts

The type of variables will be explicitly given in a typing-context. They are finite sets of (variable,type)-pairs:

$$\Gamma = \{(x, \mathsf{bool}), (y, \mathsf{bool}), (z, \mathsf{nat})\}$$

Type-Contexts

The type of variables will be explicitly given in a typing-context. They are finite sets of (variable,type)-pairs:

$$\Gamma = \{(x, \mathsf{bool}), (y, \mathsf{bool}), (z, \mathsf{nat})\}$$

we write them as

$$\Gamma = \{x : \mathsf{bool}, y : \mathsf{bool}, z : \mathsf{nat}\}$$

Type-Contexts

The type of variables will be explicitly given in a typing-context. They are finite sets of (variable,type)-pairs:

$$\Gamma = \{(x, \mathsf{bool}), (y, \mathsf{bool}), (z, \mathsf{nat})\}$$

we write them as

$$\Gamma = \{x : \mathsf{bool}, y : \mathsf{bool}, z : \mathsf{nat}\}$$

Our typing-judgement is now a 3-place relation

$$\Gamma \vdash e : T$$

Typing with Contexts

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash 0 : \mathsf{nat}}$$

$$\frac{\Gamma \vdash e : \mathsf{nat}}{\Gamma \vdash \mathsf{iszero}\ e : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{nat} \quad \Gamma \vdash e_2 : \mathsf{nat}}{\Gamma \vdash \mathsf{gr}\ e_1\ e_2 : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{nat} \quad \Gamma \vdash e_2 : \mathsf{nat}}{\Gamma \vdash \mathsf{le}\ e_1\ e_2 : \mathsf{bool}}$$

$$\frac{\Gamma \vdash e : \mathsf{nat}}{\Gamma \vdash \mathsf{succ}\ e : \mathsf{nat}}$$

Typing with Contexts

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \mathsf{if}\ e_1\ e_2\ e_3 : T}$$

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}$$

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T}$$

Typing with Contexts

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \mathsf{if}\ e_1\ e_2\ e_3 : T}$$

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}$$

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T}$$

The context must give a unique answer! E.g.:

$$\Gamma = \{(x : \mathsf{bool}), (x : \mathsf{nat})\}$$

should not be allowed.

Valid Contexts

Valid contexts are either the empty context or the ones where the domain contains only distinct variables.

$$\frac{}{\mathsf{valid}\ \varnothing}$$

$$\frac{\mathsf{valid}\ \Gamma \quad x \notin \mathsf{dom}\ \Gamma}{\mathsf{valid}\ (x : T) \cup \Gamma}$$

e.g. $\mathsf{dom}(\{x : \mathsf{bool}, y : \mathsf{bool}, z : \mathsf{nat}\}) = \{x, y, z\}$

Valid Contexts

Valid contexts are either the empty context or the ones where the domain contains only distinct variables.

$$\frac{}{\text{valid } \varnothing}$$

$$\frac{\text{valid } \Gamma \quad x \notin \text{dom } \Gamma}{\text{valid } (x : T) \cup \Gamma}$$

Now the typing-rule for variables looks as follows:

$$\frac{\text{valid } \Gamma \qquad (x : T) \in \Gamma}{\Gamma \vdash x : T}$$

Valid Contexts

Valid contexts are either the empty context or the ones where the domain contains only distinct variables.

$$\frac{}{\mathsf{valid}\ \varnothing}$$

$$\frac{\mathsf{valid}\ \Gamma \quad x \notin \mathsf{dom}\ \Gamma}{\mathsf{valid}\ (x : T) \cup \Gamma}$$

The typing-rules for true, false and 0 are:

$$\frac{\mathsf{valid}\ \Gamma}{\Gamma \vdash \mathsf{true} : \mathsf{bool}} \qquad \frac{\mathsf{valid}\ \Gamma}{\Gamma \vdash \mathsf{false} : \mathsf{bool}} \qquad \frac{\mathsf{valid}\ \Gamma}{\Gamma \vdash 0 : \mathsf{nat}}$$

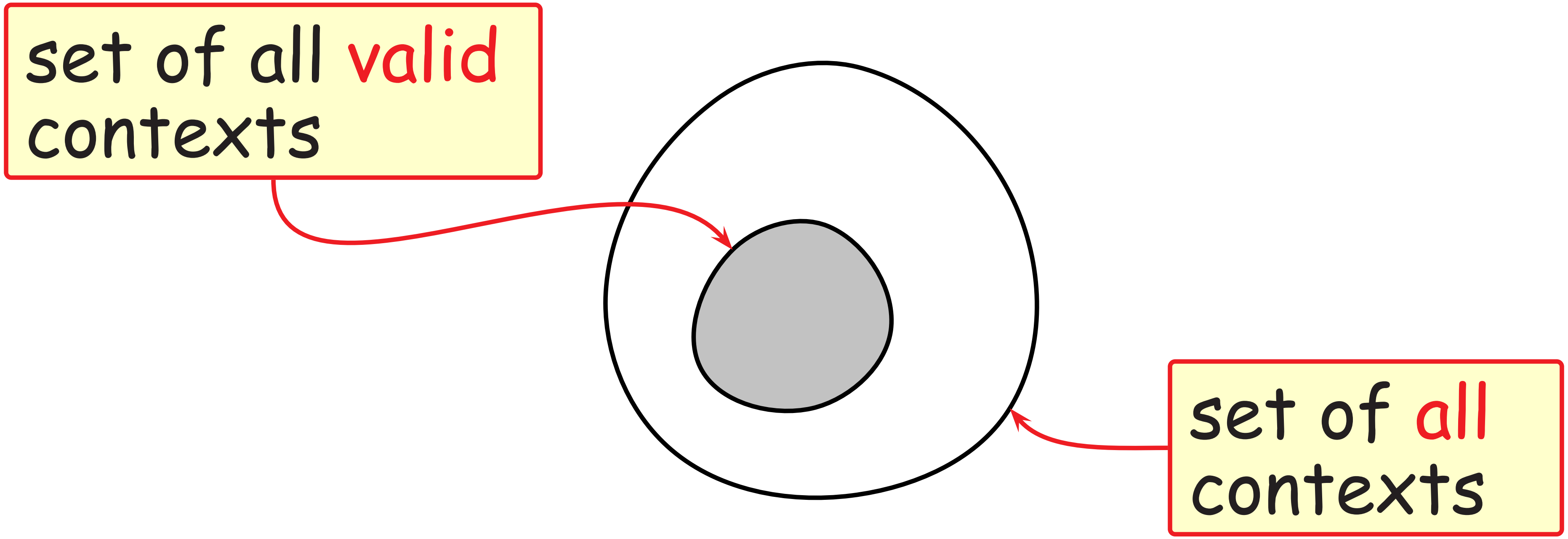
Typable

- We call an expression (term) e to be typable if there exists a Γ and a type T such that $\Gamma \vdash e : T$ can be derived.
- Not all terms are typable, e.g. for eq 0 true there does not exist such a Γ and T (according to our rules).
- We call things like:

$$\dfrac{\dfrac{(x\!:\!\mathsf{bool}) \in \{x\!:\!\mathsf{bool}\}}{\{x\!:\!\mathsf{bool}\} \vdash x : \mathsf{bool}} \quad \dfrac{}{\{x\!:\!\mathsf{bool}\} \vdash 0\!:\!\mathsf{nat}} \quad \dfrac{\dfrac{}{\{x\!:\!\mathsf{bool}\} \vdash 0\!:\!\mathsf{nat}}}{\{x\!:\!\mathsf{bool}\} \vdash \mathsf{succ}\ 0\!:\!\mathsf{nat}}}{\{x\!:\!\mathsf{bool}\} \vdash \mathsf{if}\ x\ 0\ (\mathsf{succ}\ 0) : \mathsf{nat}}$$

a derivation (in this case a type-derivation).

Inductive Definitions

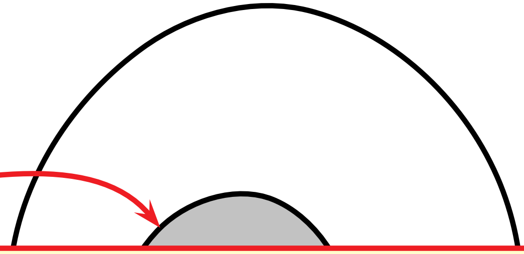


Contexts are sets of (variable,type)-pairs.

$$\frac{}{\text{valid } \varnothing} \qquad\qquad \frac{\text{valid } \Gamma \quad x \notin \text{dom } \Gamma}{\text{valid } (x : T) \cup \Gamma}$$

Inductive Definitions

set of all valid contexts



Similarly with the typing-judgement:

$$\Gamma \vdash e : T$$

and the rules we defined.

Contexts are sets of (variable,type)-pairs.

$$\frac{}{\text{valid } \varnothing} \qquad \frac{\text{valid } \Gamma \quad x \notin \text{dom } \Gamma}{\text{valid } (x : T) \cup \Gamma}$$

Inference Rules

The general pattern of an (inference) rule:

$$\frac{\text{premise}_1 \ldots \text{premise}_n \quad \text{side-conditions}}{\text{conclusion}}$$

Examples:

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \text{eq } e_1\ e_2 : \text{bool}}$$

$$\frac{\text{valid } \Gamma \quad x \notin \text{dom } \Gamma}{\text{valid } (x : T) \cup \Gamma}$$

$$\frac{}{\text{valid } \varnothing}$$

Inference Rules

The general pattern of an (inference) rule:

$$\frac{\text{premise}_1 \ldots \text{premise}_n \quad \text{side-conditions}}{\text{conclusion}}$$

Examples:

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \text{eq } e_1\ e_2 : \text{bool}}$$

$$\frac{\text{valid } \Gamma \quad x \notin \text{dom } \Gamma}{\text{valid } (x : T) \cup \Gamma}$$

$$\frac{}{\text{valid } \varnothing}$$

An axiom is an inference rule without premises (it can have side-conditions), e.g:

$$\frac{\text{valid } \Gamma \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}$$

Induction Principles

Remember the general pattern of a rule is:

$$\frac{\text{premise}_1 \ldots \text{premise}_n \quad \text{side-conditions}}{\text{conclusion}}$$

Induction Principles

Remember the general pattern of a rule is:

$$\frac{\text{premise}_1 \ldots \text{premise}_n \quad \text{side-conditions}}{\text{conclusion}}$$

We can show that a property P holds for all elements given by rules, by

- showing that the property holds for the axioms (we can assume the side-conditions)
- holds for the conclusion of all other rules, assuming it holds already for the premises (we can also assume the side-conditions)

For Example

- We want to show that a property $P\ \Gamma\ e\ T$ holds for all $\Gamma \vdash e : T$. That means we want to show

$$\Gamma \vdash e : T \Rightarrow P\ \Gamma\ e\ T$$

For Example

- We want to show that a property $P\ \Gamma\ e\ T$ holds for all $\Gamma \vdash e : T$. That means we want to show

$$\Gamma \vdash e : T \Rightarrow P\ \Gamma\ e\ T$$

- For every rule

$$\frac{\text{premise}_1 \ldots \text{premise}_n \quad \text{side-conditions}}{\text{conclusion}}$$

"P prem$_1$" $\wedge \ldots \wedge$ "P prem$_n$" $\wedge$ side-cond's $\Rightarrow$ "P concl"

For Example

- So for the gr-rule

$$\frac{\Gamma \vdash e_1 : \mathsf{nat} \quad \Gamma \vdash e_2 : \mathsf{nat}}{\Gamma \vdash \mathsf{gr}\ e_1\ e_2 : \mathsf{bool}}$$

$$P\ \Gamma\ e_1\ \mathsf{nat} \wedge P\ \Gamma\ e_2\ \mathsf{nat} \Rightarrow P\ \Gamma\ (\mathsf{gr}\ e_1\ e_2)\ \mathsf{bool}$$

For Example

- So for the gr-rule

$$\frac{\Gamma \vdash e_1 : \mathsf{nat} \quad \Gamma \vdash e_2 : \mathsf{nat}}{\Gamma \vdash \mathsf{gr}\ e_1\ e_2 : \mathsf{bool}}$$

$$P\ \Gamma\ e_1\ \mathsf{nat} \wedge P\ \Gamma\ e_2\ \mathsf{nat} \Rightarrow P\ \Gamma\ (\mathsf{gr}\ e_1\ e_2)\ \mathsf{bool}$$

- and for the true-axiom

$$\frac{\mathsf{valid}\ \Gamma}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}$$

$$\mathsf{valid}\ \Gamma \Rightarrow P\ \Gamma\ \mathsf{true}\ \mathsf{bool}$$

Induction in Action

- Let's show a concrete property:

$$P\ \Gamma\ e\ T \stackrel{\text{def}}{=} \text{valid}\ \Gamma$$

- That means we want to show: If $\Gamma \vdash e : T$ then $\text{valid}\ \Gamma$, or

$$\Gamma \vdash e : T \Rightarrow \text{valid}\ \Gamma$$

- Proof by induction over the rules of $\Gamma \vdash e : T$:

  1) we have to show P for the axioms, and
  2) then for the other rules

1. Axioms

$$\frac{\text{valid } \Gamma}{\Gamma \vdash \text{true} : \text{bool}} \qquad \frac{\text{valid } \Gamma}{\Gamma \vdash \text{false} : \text{bool}} \qquad \frac{\text{valid } \Gamma}{\Gamma \vdash 0 : \text{nat}}$$

$$\frac{\text{valid } \Gamma \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}$$

1. Axioms

$$\frac{\text{valid }\Gamma}{\Gamma \vdash \text{true} : \text{bool}} \qquad \frac{\text{valid }\Gamma}{\Gamma \vdash \text{false} : \text{bool}} \qquad \frac{\text{valid }\Gamma}{\Gamma \vdash 0 : \text{nat}}$$

$$\frac{\text{valid }\Gamma \quad (x : T) \in \Gamma}{\Gamma \vdash x : T}$$

$$\text{"side-cond's"} \Rightarrow \text{"}P\,\text{concl"}$$

$$\text{valid }\Gamma \Rightarrow \text{valid }\Gamma$$

$$\text{valid }\Gamma \wedge (x : T) \in \Gamma \Rightarrow \text{valid }\Gamma$$

2. Rules

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \mathsf{if}\ e_1\ e_2\ e_3 : T}$$

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}$$

2. Rules

$$\frac{\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \mathsf{if}\ e_1\ e_2\ e_3 : T}$$

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}}$$

"P prem's" $\wedge$ "side-cond's" $\Rightarrow$ "Pconcl"

valid Γ $\wedge$ valid Γ $\wedge$ valid Γ $\Rightarrow$ valid Γ

valid Γ $\wedge$ valid Γ $\Rightarrow$ valid Γ

2. Rules

$$\Gamma \vdash e_1 : \mathsf{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T$$

If we go through all cases, we proved:

Whenever $\Gamma \vdash e : T$ then valid Γ.

OK that was simple.

$$\Gamma \vdash \mathsf{eq}\ e_1\ e_2 : \mathsf{bool}$$

"P prem's" $\wedge$ "side-cond's" $\Rightarrow$ "Pconcl"

valid Γ $\wedge$ valid Γ $\wedge$ valid Γ $\Rightarrow$ valid Γ

valid Γ $\wedge$ valid Γ $\Rightarrow$ valid Γ

2. Rules

$$\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T$$

If we go through all cases, we proved:

Whenever $\Gamma \vdash e : T$ then valid Γ.

OK that was simple.

$$\Gamma \vdash \text{eq}\ e_1\ e_2 : \text{bool}$$

But one has to know a hammer, before one can crack a nut. ;o)

valid $\Gamma \wedge$ valid $\Gamma \Rightarrow$ valid Γ

Structural Induction

$$
\begin{array}{l}
\forall x.\, P\, x \\
P\,\mathsf{true} \\
P\,\mathsf{false} \\
\forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow \\
\forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow \\
\forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow \\
\forall e_1\, e_2\, e_3.\, P\, e_1 \wedge P\, e_2 \wedge \\
P\, 0 \\
\forall e.\, P\, e \Rightarrow P\, (\mathsf{succ}\, e) \\
\forall e.\, P\, e \Rightarrow P\, (\mathsf{iszero}\, e) \\
\hline
\forall e.\, P\, e
\end{array}
$$

$$
\begin{array}{rcl}
e & ::= & x \\
 & \mid & \mathsf{true} \\
 & \mid & \mathsf{false} \\
 & \mid & \mathsf{gr}\; e\; e \\
 & \mid & \mathsf{le}\; e\; e \\
 & \mid & \mathsf{eq}\; e\; e \\
 & \mid & \mathsf{if}\; e\; e\; e \\
 & \mid & 0 \\
 & \mid & \mathsf{succ}\; e \\
 & \mid & \mathsf{iszero}\; e
\end{array}
$$

Structural Induction

$$\dfrac{\begin{array}{l} \forall x.\, P\, x \\ P\ \mathsf{true} \\ P\ \mathsf{false} \\ \forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow P\, (\mathsf{gr}\ e_1\, e_2) \\ \forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow P\, (\mathsf{le}\ e_1\, e_2) \\ \forall e_1\, e_2.\, P\, e_1 \wedge P\, e_2 \Rightarrow P\, (\mathsf{eq}\ e_1\, e_2) \\ \forall e_1\, e_2\, e_3.\, P\, e_1 \wedge P\, e_2 \wedge P\, e_3 \Rightarrow P\, (\mathsf{if}\ e_1\, e_2\, e_3) \\ P\, 0 \\ \forall e.\, P\, e \Rightarrow P\, (\mathsf{succ}\ e) \\ \forall e.\, P\, e \Rightarrow P\, (\mathsf{iszero}\ e) \end{array}}{\forall e.\, P\, e}$$

More Next Week

- Slides at the end of

  http://www4.in.tum.de/lehre/vorlesungen/types/WS0607/

  There is also an appraisal form where you can complain anonymously.

- You can say whether the lecture was too easy, too quiet, too hard to follow, too chaotic and so on. You can also comment on things I should repeat.