Type Systems

Lecture 1 Oct. 20th, 2004
Sebastian Maneth

http://lampwww.epfl.ch/teaching/typeSystems/2004

Today

1. Organizational Matters
2. What is this course about?
3. Where do “types” come from?
4. Def. of the small language Expr. Its syntax and semantics.
5. Structural Induction on Expr’s

1. Organizational Matters

Lectures:	Exercises (lab):
We 13:15-15:00, INM203	We 15:15-17:00, INR 331
Sebastian Maneth BC360, 021-69 31226	Burak Emir INR320, 021-69 36867
(last 3 lectures by Martin Odersky)	

To get credits you have to:

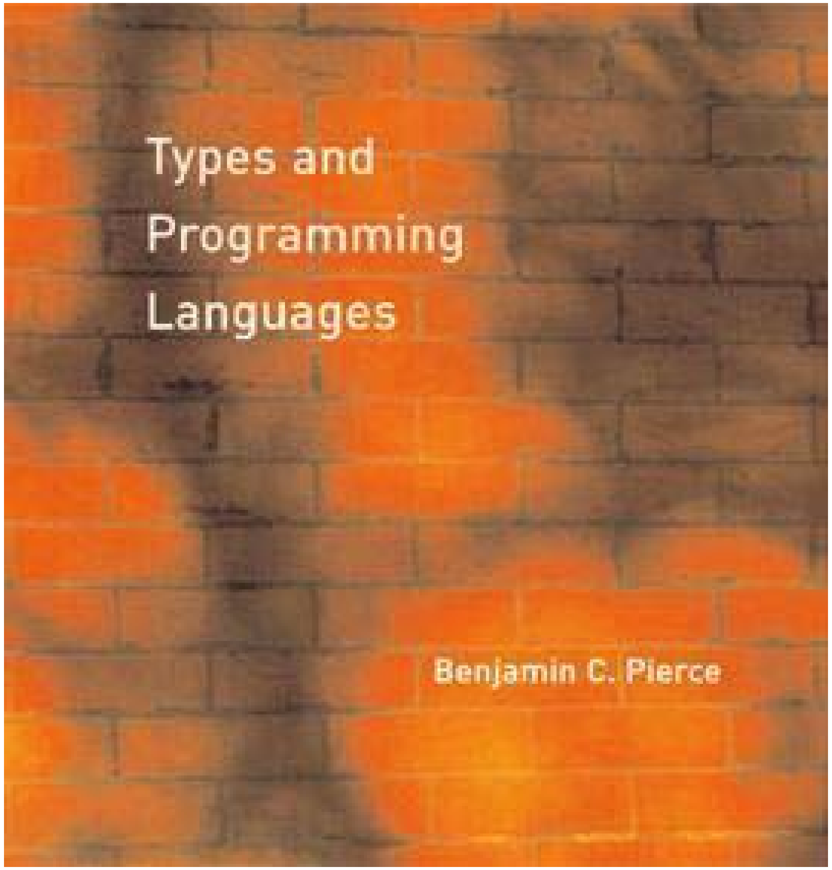
1/3 { → 1-2 written assignments
→ one programming assignment

2/3 → oral examination

1. Organizational Matters

Course Book: Benjamin Pierce, “Types and Programming Languages”

MIT Press, 2002.



We will strictly follow this book!

So: Good to buy it!

Type Systems for Programming Languages

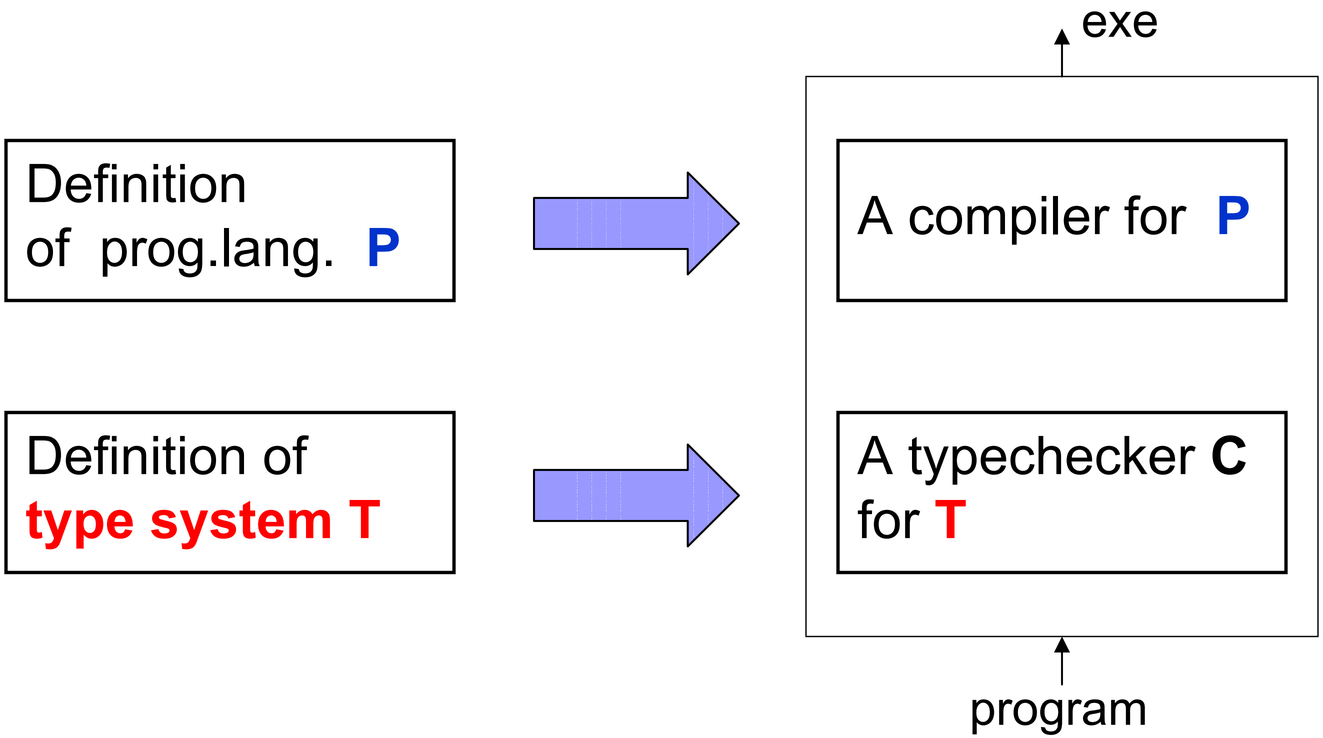
What for ??

→ to prevent **execution errors**.

A PL in which all well-typed programs are free of execution errors

is called **type sound**.

Type Systems for Programming Languages



exe

Definition of prog.lang. **P**

A compiler for **P**

Definition of **type system T**

A typechecker **C** for **T**

program

→ is (**P**, **T**) type sound?

→ is **T** decidable?

→ does **C** correctly implement **T**?

What you will learn in this course:

- how to **define** a type system **T** (to allow for unambiguous implementations)

- how to formally **prove** that (**P**, **T**) is type sound

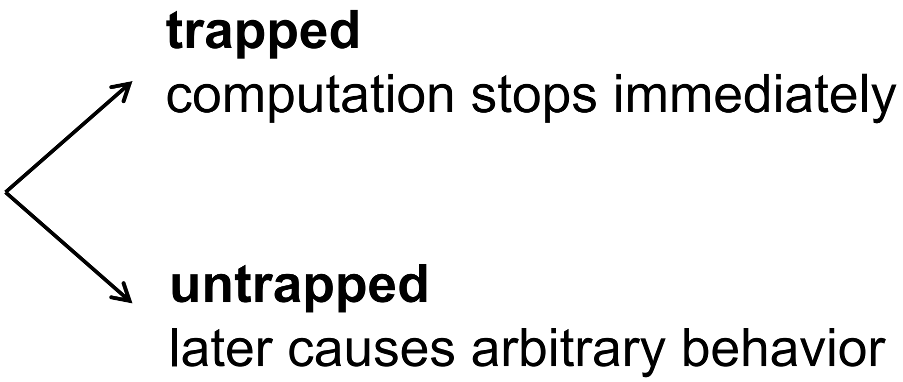
- how to **implement** a typechecker for **T**

Type Systems in Programming Languages

What for ??

→ to prevent **execution errors**.

Execution Errors



trapped
computation stops immediately

untrapped
later causes arbitrary behavior

examples:

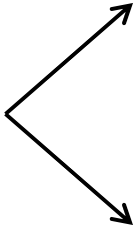
- division by zero
- accessing an illegal addr.

- jump to a wrong addr.
- accessing past the end of an array

A program is **SAFE** if it does not have untrapped errors.

A PL is **SAFE** if all its programs are safe.

Execution Errors



trapped
computation stops immediately

untrapped
later causes arbitrary behavior

examples:

- division by zero
- accessing an illegal addr.

- jump to a wrong addr.
- accessing past the end of an array

A program is **SAFE** if it does not have untrapped errors.

A PL is **SAFE** if all its programs are.

$\underbrace{\text{trapped + some “forbidden” untrapped errors}}$:= well-behaved

What is a TYPE, in our context?

A **type** is an upper bound of the **range of values** that a **program variable** can assume during execution.

e.g. if x has type Boolean, then in all runs it should only take one of the values true / false.

→ not(x) has a meaning in every run

PLs in which variables can be given nontrivial types are called **TYPED languages**.

safe/unsafe and typed/untyped

	typed	untyped
safe	ML, Java	LISP
unsafe	C	Assembler

safety ⇒ integrity of run-time structures

⇒ enables garbage collection

⇓

saves code size / develop. time

(price: performance)

safe/unsafe and typed/untyped

	typed	untyped
safe	ML, Java	LISP
unsafe	C	Assembler

safety ⇒ integrity of run-time structures

⇒ enables garbage collection

⇓

saves code size / develop. time

(price: performance)

SECURITY vs. PERFORMANCE

```
var x : Boolean
    ⋮
        x := 10;
```

typechecker should

complain!

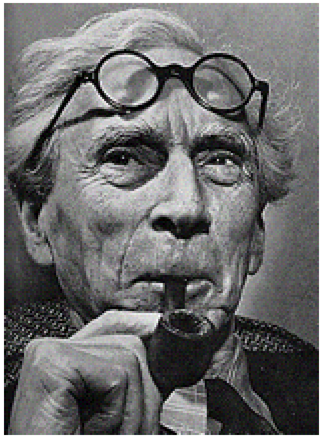
caveat: of course no one knows if this line will ever be executed!
… but … it just not SAFE to have it.

should **not** be allowed to write such a program: it has **no meaning**!

TYPE SYSTEMS are there to PROTECT YOU from making

stupid (obvious) mistakes.

Type Theory is much older than PLs!



Bertrand Russell (1872-1970)

1901 Russell's Paradox Let $P = \{ Q \in \text{sets} \mid Q \notin Q\}$

then: $P \in P \Leftrightarrow P \notin P$

$\Rightarrow$ Naive set theory is inconsistent!

$\Rightarrow$ MUST eliminate self-referential defs.
to make set theory consistent

HOW?

1903 define a hierarchy of types: individuals, sets, sets of set, etc.

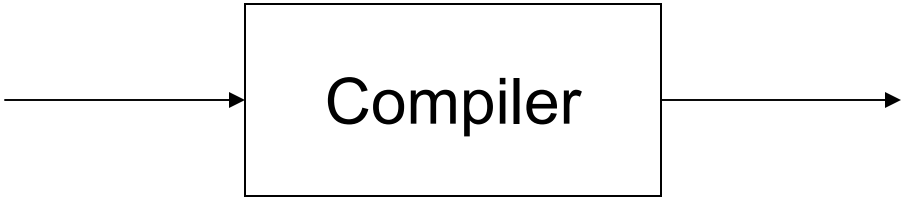
Any well defined set can only have elements from lower levels.

Course Outline

- today: Intro, Arithm. Expressions, Induction, Evaluation → LAB1
- next: (untyped) Lambda-Calculus → LAB2 untyped λ-evaluator
- 3rd: Simply-Typed Lambda-Calculus → LAB3 simply typed w. let/fix
- 4rd: Simple Extensions, Subtyping → LAB4 subtyping on records
- 5th: Subtyping, Featherweight Java → LAB5
- 6th: Recursive Types I
- 7th: Recursive Types II
- 8th: Polymorphism I
- 9th: Polymorphism II
- 10th: Bounded Quantification
- 11-13th: Scala's Type System (by Martin Odersky)

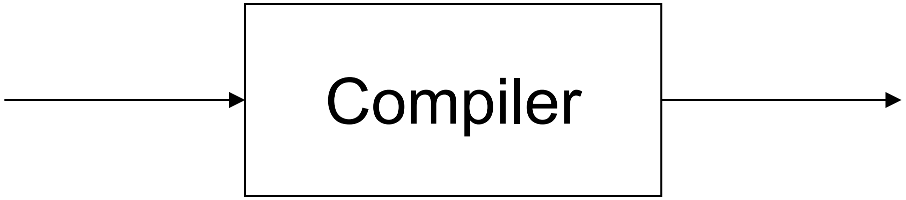
Syntax and Semantics of PLs

1960 Irons, Syntax-Directed Compiler for ALGOL 60



Syntax and Semantics of PLs

1960 Irons, Syntax-Directed Compiler for ALGOL 60

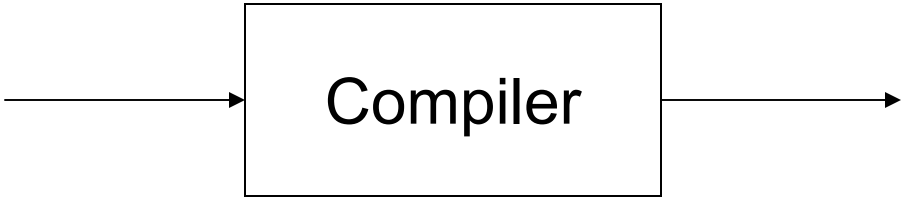


Compiler

Defining | Translating

Syntax and Semantics of PLs

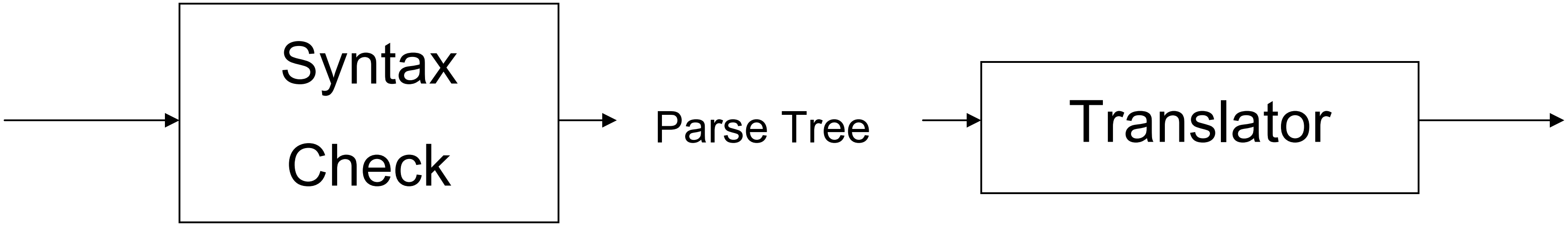
1960 Irons, Syntax-Directed Compiler for ALGOL 60



Compiler

Defining | Translating

1966 Younger, O(n^3) Parsing of **Context-Free Grammars**



Syntax Check

Parse Tree

Translator

Syntax and Semantics of PLs

Until today, **EBNF** (ext. cf. grammar) is used to describe the **syntax of a programming language**.

Example: Arithmetic Expressions

```
Expr  ::=  true | false | zero
Expr  ::=  if Expr then Expr else Expr
Expr  ::=  succ Expr
Expr  ::=  pred Expr
Expr  ::=  isZero Expr
```

Derivable Expressions:

→ pred succ zero
→ if isZero pred succ zero then zero else true
→ if zero then true else false

Syntax and Semantics of PLs

Until today, **EBNF** (ext. cf. grammar) is used to describe

the **syntax of a programming language**.

Example: Arithmetic Expressions

```
Expr  ::=  true | false | zero
Expr  ::=  if Expr then Expr else Expr
Expr  ::=  succ (Expr)
Expr  ::=  pred (Expr)
Expr  ::=  isZero (Expr)
```

Derivable Expressions:

→ pred (succ (zero))
→ if isZero (pred (succ (zero))) then zero else true
→ if zero then true else false

Syntax and Semantics of PLs

Until today, **EBNF** (ext. cf. grammar) is used to describe

the **syntax of a programming language**.

Example: Arithmetic Expressions

```
Expr  ::=  true | false | zero
Expr  ::=  if Expr then Expr else Expr
Expr  ::=  succ (Expr)
Expr  ::=  pred (Expr)
Expr  ::=  isZero (Expr)
```

Derivable Expressions:

→ pred (succ (zero))
→ if isZero (pred (succ (zero))) then zero else true
→ if zero then true else false

semantics??

Syntax and Semantics of PLs

Alternative Formalism: **Inference Rules**

The set of expressions is the smallest set E such that:

$$\text{true} \in E \qquad \text{false} \in E \qquad \text{zero} \in E$$

$$\frac{t_1 \in E}{\text{succ } t_1 \in E} \qquad \frac{t_1 \in E}{\text{pred } t_1 \in E} \qquad \frac{t_1 \in E}{\text{isZero } t_1 \in E}$$

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

Syntax and **Semantics** of PLs

1. **Operational Semantics**: behavior defined in terms of abstract machines

2. **Denotational Semantics**: maps programs by an interpretation function into a collection of semantic domains (such as, e.g., numbers, functions, etc.)

3. **Axiomatic Semantics**: proves properties of a program by applying laws about program behavior (e.g., given that properties P hold before a statement, what properties Q hold after executing it?)

Syntax and **Semantics** of PLs

1. **Operational Semantics**: behavior defined in terms of abstract machines

2. **Denotational Semantics**: maps programs by an interpretation function into a collection of semantic domains (such as, e.b., numbers, functions, etc)

3. **Axiomatic Semantics**: proves properties of a program by applying laws about program behavior (e.g., given that properties P hold before a statement, what properties Q hold after executing it?)

Semantics of Expr

Expr ::= true | false | zero

Expr ::= if Expr then Expr else Expr

Expr ::= succ (Expr)

Expr ::= pred (Expr)

Expr ::= isZero (Expr)

Val ::= true | false | NVal

NVal ::= zero | succ NVal

Evaluation Relation → on Expr's

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

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

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

Semantics of Expr

Expr ::= true | false | zero
Expr ::= if Expr then Expr else Expr
Expr ::= succ (Expr)
Expr ::= pred (Expr)
Expr ::= isZero (Expr)

Val ::= true | false | NVal
NVal ::= zero | succ NVal

Evaluation Relation → on Expr's

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

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

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

$$\frac{t_1 \rightarrow t_1'}{\text{succ } t_1 \rightarrow \text{succ } t_1'} \qquad \frac{t_1 \rightarrow t_1'}{\text{pred } t_1 \rightarrow \text{pred } t_1'} \qquad \frac{t_1 \rightarrow t_1'}{\text{isZero } t_1 \rightarrow \text{isZero } t_1'}$$

$$\text{pred zero} \rightarrow \text{zero} \qquad \text{isZero zero} \rightarrow \text{true}$$

$$\text{pred succ } nv_1 \rightarrow nv_1 \qquad \text{isZero succ } nv_1 \rightarrow \text{false}$$

Semantics of Expr

Example: if isZero pred succ pred zero then zero else succ zero

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

$$\frac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \qquad \frac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \qquad \frac{t_1 \to t_1'}{\text{isZero } t_1 \to \text{isZero } t_1'}$$

$$\text{pred zero} \to \text{zero} \qquad \text{isZero zero} \to \text{true}$$

$$\text{pred succ } nv_1 \to nv_1 \qquad \text{isZero succ } nv_1 \to \text{false}$$

Semantics of Expr

Example: if isZero pred succ $\overset{\text{redex}}{\boxed{\text{pred zero}}}$ then zero else succ zero

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

$$\frac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \qquad \frac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \qquad \frac{t_1 \to t_1'}{\text{isZero } t_1 \to \text{isZero } t_1'}$$

$\boxed{\text{pred zero} \to \text{zero}}$ isZero zero → true

pred succ $nv_1 \to nv_1$ isZero succ nv_1 → false

Semantics of Expr

Example: $\text{if isZero pred succ } \overset{\text{redex}}{\boxed{\text{pred zero}}} \text{ then zero else succ zero}$

$\rightarrow$ $\text{if isZero pred succ } \quad \text{zero} \quad \text{ then zero else succ zero}$

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

$$\frac{t_1 \rightarrow t_1'}{\text{succ } t_1 \rightarrow \text{succ } t_1'} \qquad \frac{t_1 \rightarrow t_1'}{\text{pred } t_1 \rightarrow \text{pred } t_1'} \qquad \frac{t_1 \rightarrow t_1'}{\text{isZero } t_1 \rightarrow \text{isZero } t_1'}$$

$\text{pred zero} \rightarrow \text{zero}$ $\qquad$ $\text{isZero zero} \rightarrow \text{true}$

$\text{pred succ } nv_1 \rightarrow nv_1$ $\qquad$ $\text{isZero succ } nv_1 \rightarrow \text{false}$

Semantics of Expr

Example: $\text{if isZero pred succ } \boxed{\text{pred zero}} \text{ then zero else succ zero}$

redex

$\to$ $\text{if isZero } \boxed{\text{pred succ zero}} \text{ then zero else succ zero}$

$$\boxed{\dfrac{t_1 \to t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \to \text{if } t_1' \text{ then } t_2 \text{ else } t_3 \text{ E}}}$$

$$\dfrac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \qquad \dfrac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \qquad \boxed{\dfrac{t_1 \to t_1'}{\text{isZero } t_1 \to \text{isZero } t_1'}}$$

$\text{pred zero} \to \text{zero}$

$\text{isZero zero} \to \text{true}$

$\boxed{\text{pred succ } nv_1 \to nv_1}$

$\text{isZero succ } nv_1 \to \text{false}$

Semantics of Expr

Example: $\text{if isZero pred succ } \boxed{\text{pred zero}} \text{ then zero else succ zero}$

redex

$\to$ $\text{if isZero } \boxed{\text{pred succ zero}} \text{ then zero else succ zero}$

$\to$ $\text{if isZero zero then zero else succ zero}$

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

$$\frac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \qquad \frac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \qquad \frac{t_1 \to t_1'}{\text{isZero } t_1 \to \text{isZero } t_1'}$$

$\text{pred zero} \to \text{zero}$ $\qquad$ $\text{isZero zero} \to \text{true}$

$\text{pred succ } nv_1 \to nv_1$ $\qquad$ $\text{isZero succ } nv_1 \to \text{false}$

Semantics of Expr

Example: if isZero pred succ $\boxed{\text{pred zero}}$ then zero else succ zero

→ if isZero $\boxed{\text{pred succ zero}}$ then zero else succ zero

redex

→ if $\boxed{\text{isZero zero}}$ then zero else succ zero

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

$$\frac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \qquad \frac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \qquad \frac{t_1 \to t_1'}{\text{isZero } t_1 \to \text{isZero } t_1'}$$

pred zero → zero $\qquad$ $\boxed{\text{isZero zero} \to \text{true}}$

pred succ $nv_1 \to nv_1$ $\qquad$ isZero succ $nv_1 \to$ false

Semantics of Expr

Example: if isZero pred succ $\boxed{\text{pred zero}}$ then zero else succ zero

→ if isZero $\boxed{\text{pred succ zero}}$ then zero else succ zero

redex

→ if $\boxed{\text{isZero zero}}$ then zero else succ zero

→ if true then zero else succ zero

$$\frac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \qquad \frac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \qquad \frac{t_1 \to t_1'}{\text{isZero } t_1 \to \text{isZero } t_1'}$$

pred zero → zero

pred succ nv_1 → nv_1

isZero zero → true

isZero succ nv_1 → false

Semantics of Expr

Example: $\text{if isZero pred succ } \boxed{\text{pred zero}} \text{ then zero else succ zero}$

$\to \quad \text{if isZero } \boxed{\text{pred succ zero}} \text{ then zero else succ zero}$

$\to \quad \text{if } \boxed{\text{isZero zero}} \text{ then zero else succ zero}$

redex

$\to \quad \boxed{\text{if true then zero else succ zero}}$

$$\frac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \qquad \frac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \qquad \frac{t_1 \to t_1'}{\text{isZero } t_1 \to \text{isZero } t_1'}$$

$\text{pred zero} \to \text{zero}$ $\quad \boxed{\text{if true then } t_2 \text{ else } t_3 \to t_2}$

$\text{pred succ } nv_1 \to nv_1$ $\quad \text{isZero succ } nv_1 \to \text{false}$ $\quad \text{isZero zero} \to \text{true}$

Semantics of Expr

Example: $\text{if isZero pred succ } \boxed{\text{pred zero}} \text{ then zero else succ zero}$

$\to \quad \text{if isZero } \boxed{\text{pred succ zero}} \text{ then zero else succ zero}$

$\to \quad \text{if } \boxed{\text{isZero zero}} \text{ then zero else succ zero}$

redex

$\to \quad \boxed{\text{if true then zero else succ zero}}$

$\to \quad \text{zero}$

$$\frac{t_1 \to t_1'}{\text{succ } t_1 \to \text{succ } t_1'} \qquad \frac{t_1 \to t_1'}{\text{pred } t_1 \to \text{pred } t_1'} \qquad \frac{t_1 \to t_1'}{\text{isZero } t_1 \to \text{isZero } t_1'}$$

$\text{pred zero} \to \text{zero}$

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

$\text{pred succ } nv_1 \to nv_1$

$\text{isZero succ } nv_1 \to \text{false}$

$\text{isZero zero} \to \text{true}$

Induction on the Structure of Expr's

The set of expressions is the smallest set E such that:

1. true, false, zero $\in$ E
2. if t_1, t_2, $t_3 \in$ E, then succ t_1, pred t_1, isZero $t_1 \in$ E
   and if t_1 then t_2 else $t_3 \in$ E

inductive definition

→ we can define / proof things about Expr's by induction!

Example: for any Expr t define its **size** as

1. if t = true | false | zero then **size**(t) = 0
2. if t = succ t_1 | pred t_1 | isZero t_1 then **size**(t) = **size**(t_1) + 1
   if t = if t_1 then t_2 else t_3 then **size**(t) = **size**(t_1) + **size**(t_2) + **size**(t_3) + 1

Proof by Induction on the Structure of Expr's

Theorem. → is deterministic: if t → t' and t → t'' then t' = t''

Proof. by induction on the structure of t

1. if t = true | false | zero then t' = t'' = t

2. if $t = \text{succ } t_1$ then

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

only rule for succ(..)

Proof by Induction on the Structure of Expr's

Theorem. → is deterministic: if t → t' and t → t'' then t' = t''

Proof. by induction on the structure of t

1. if t = true | false | zero then t' = t'' = t

2. if t = succ t_1 then t' = succ t_1' and t'' = succ t_1''
   for t_1', t_1'' with $t_1 \rightarrow t_1'$ and $t_1 \rightarrow t_1''$

Proof by Induction on the Structure of Expr’s

Theorem. → is deterministic: if t → t’ and t → t’’ then t’ = t’’

Proof. by induction on the structure of t

1. if t = true | false | zero then t’ = t’’ = t

2. if t = succ t_1 then t’ = succ t_1’ and t’’ = succ t_1’’
   for t_1’, t_1’’ with $\underbrace{t_1 \to t_1' \text{ and } t_1 \to t_1''}_{\text{by induction } t_1' = t_1''}$

Proof by Induction on the Structure of Expr’s

Theorem. → is deterministic: if $t \to t'$ and $t \to t''$ then $t' = t''$

Proof. by induction on the structure of t

1. if $t =$ true | false | zero then $t' = t'' = t$

2. if $t =$ succ t_1 then $t' =$ succ t_1' and $t'' =$ succ t_1''
   for t_1', t_1'' with $\underbrace{t_1 \to t_1' \text{ and } t_1 \to t_1''}_{\text{by induction } t_1' = t_1''}$

   Thus, also $t' = t''$.

Proof by Induction on the Structure of Expr’s

Theorem. → is deterministic: if t → t’ and t → t’’ then t’ = t’’

Proof. by induction on the structure of t

1. if t = true | false | zero then t’ = t’’ = t

2. if t = pred t_1 then

   if t_1 = succ t_{11} then t’ = t’’ = t_{11}

   because $\boxed{\text{pred succ } nv_1 \rightarrow nv_1}$ is **only** rule applicable.

Proof by Induction on the Structure of Expr's

Theorem. → is deterministic: if t → t' and t → t'' then t' = t''

Proof. by induction on the structure of t

1. if t = true | false | zero then t' = t'' = t

2. if t = pred t_1 then

   if t_1 = succ t_{11} then t' = t'' = t_{11}

   because $\boxed{\text{pred succ } nv_1 \rightarrow nv_1}$ is **only** rule applicable.

   otherwise t' = pred t_1' and t'' = pred t_1''
   with $t_1 \rightarrow t_1$' and $t_1 \rightarrow t_1$''

Proof by Induction on the Structure of Expr’s

Theorem. → is deterministic: if $t \rightarrow t'$ and $t \rightarrow t''$ then $t' = t''$

Proof. by induction on the structure of t

1. if $t =$ true | false | zero then $t' = t'' = t$

2. if $t =$ pred t_1 then

   if $t_1 =$ succ t_{11} then $t' = t'' = t_{11}$

   because $\boxed{\text{pred succ } nv_1 \rightarrow nv_1}$ is **only** rule applicable.

   otherwise $t' =$ pred t_1' and $t'' =$ pred t_1''

   with $\underbrace{t_1 \rightarrow t_1' \text{ and } t_1 \rightarrow t_1''}_{\text{by induction } t_1' = t_1''}$

Thus, also $t' = t''$.

Proof by Induction on the Structure of Expr’s

Theorem. → is deterministic: if t → t’ and t → t’’ then t’ = t’’

Proof. by induction on the structure of t

1. <u>if</u> t = true | false | zero <u>then</u> t’ = t’’ = t

2. <u>if</u> t = if t_1 then t_2 else t_3 <u>then</u>

   <u>if</u> t_1 = true <u>then</u> t’ = t’’ = t_2

   <u>if</u> t_1 = false <u>then</u> t’ = t’’ = t_3

Proof by Induction on the Structure of Expr’s

Theorem. → is deterministic: if t → t’ and t → t’’ then t’ = t’’

Proof. by induction on the structure of t

1. if t = true | false | zero then t’ = t’’ = t

2. if t = if t_1 then t_2 else t_3 then

   if t_1 = true then $t' = t'' = t_2$

   if t_1 = false then $t' = t'' = t_3$

   otherwise t' = if t_1' then t_2 else t_3 and
   t'' = if t_1'' then t_2 else t_3

   with $\underbrace{t_1 \to t_1' \text{ and } t_1 \to t_1''}_{\text{by induction } t_1' = t_1''}$

Thus, also t’ = t’’.

Questions:

1. Is → still deterministic if we add the new rule

   $\text{succ pred nv}_1 \rightarrow \text{nv}_1$

   Which rule must be removed now, to keep a sane semantics?

2. What if redexes can be chosen freely? Is → still determin.?

   (i.e., rules can be applied to arbitrary sub-Expr's)

   Is → confluent? Is it terminating?

if
$$\begin{tikzcd} & t_1 \\ t \arrow[ur] \arrow[dr] & \\ & t_2 \end{tikzcd}$$
then there is a t' such that
$$\begin{tikzcd} t_1 \arrow[r] & \cdots \arrow[dr] & \\ & & t' \\ t_2 \arrow[r] & \cdots \arrow[ur] & \end{tikzcd}$$

Summary

→ we have defined the **syntax** of the small language called Expr.

→ we have given a **semantics** to Expr's by means of an evaluation relation.

→ we have proved by **induction** that for every Expr there is at most one other Expr that can be derived by the evaluation relation.

Next Lecture

How to define a small language for defining **functions**?

→ function definition and application: the lambda-calculus