Fundamentals of MCS (CS)

Grammar, Semantics, and Typing Relation

2014.11.26 and 2014.12.1 Ken Wakita

Outline

- Grammar – Structure of the Program
- Evaluation – Dynamic Semantics of the Program
- Type System and Typing – Static Semantics of the Program

A Lesson from Type Analysis

Types in Computer Science

- A **type system** is a tractable syntactic method for **proving** the absence of certain **program** behaviors by classifying phrases accord to the kinds of values they compute.
  - The notion of types appear in broader field of study: logic, mathematics, and philosophy.

What is a type system?

- A **type system** is a tractable syntactic method for proving the absence of certain program behaviors by **classifying phrases accord to the kinds of values they compute**.
    - a type system can be regarded as calculating a kind of **static approximation to the run-time** behaviors of the term in a program.

What is a type system?

- a **type system** can be regarded as calculating a kind of **static approximation to the run-time** behaviors of the term in a program.

- **Example (Fibonacci series)**

$f_1 = 1, f_2 = 2;$
while $f_1 < 100$ **do**
  $(f_1, f_2) := (f_2, f_1+f_2)$
done

$f_1 =[1, 2, 3, 5, 8, 13, 21, 34, 55, 89]$

What is a type system?

- A type system is a tractable syntactic method for **proving the absence of certain program behaviors** by classifying phrases accord to the kinds of values they compute.

  function f() = a_complex_test_that_always_gives_**true**

  if f() **then** 5 **else** e^{-2i} (“apple” / π) $\Rightarrow$ ***type error***

What is a type system?

- A type system is a **tractable** syntactic method for proving the absence of certain program behaviors by classifying phrases accord to the kinds of values they compute.
  - Type-checkers are typically built into compilers and linkers and do their job automatically.

What is a type system

- we come back to this topic next week

Grammar

Syntax of Arithmetic Expressions

t ::=	terms of arithmetic expressions
| true	constant true
| false	constant false
| if t then t else t	conditional
| 0	constant zero
| succ t	successor of t (i.e., t + 1)
| pred t	predecessor of t (i.e., t - 1)
| iszero t	if t is zero?

Examples

- if false then 0 else succ 0
  $\Rightarrow$ 1
- succ (succ (succ 0))
  $\Rightarrow$ 3
- iszero 0
  $\Rightarrow$ true
- iszero (succ 0)
  $\Rightarrow$ false
- iszero (pred (succ 0))
  $\Rightarrow$ true

Inductive definition on terms
Constants in arithmetic expressions

t ::=		terms of arithmetic expressions
	| true	**constant** true
	| false	**constant** false
	| if t_1 then t_2 else t_3	conditional
	| 0	**constant** zero
	| succ t	successor of t (t + 1)
	| pred t	predecessor of t (t - 1)
	| iszero t	if t is zero?

Inductive definition on terms
Constants in arithmetic expressions

Consts(t)	the set of constants in AEs
Consts(true)	= { true }
Consts(false)	= { false }
if t_1 then t_2 else t_3	conditional
Consts(0)	= { 0 }
succ t	successor of t (t + 1)
pred t	predecessor of t (t - 1)
iszero t	if t is zero?

Inductive definition on terms
Constants in arithmetic expressions

Consts(t)	the set of constants in AEs
Consts(true)	= { true }
Consts(false)	= { false }
if t_1 then t_2 else t_3	conditional
Consts(0)	= { zero }
Consts(succ t)	Consts(t)
Consts(pred t)	Consts(t)
Consts(iszero t)	Consts(t)

Inductive definition on terms
Constants in arithmetic expressions

Consts(t)	the set of constants in AEs
Consts(true)	= { true }
Consts(false)	= { false }
Consts(if t_1 then t_2 else t_3)	= Consts(t_1) ∪ Consts(t_2) ∪ Consts(t_3)
Consts(0)	= { zero }
Consts(succ t)	Consts(t)
Consts(pred t)	Consts(t)
Consts(iszero t)	Consts(t)

Inductive definition on terms
The **depth** of arithmetic expressions

t ::=	terms of arithmetic expressions
true	constant true
false	constant false
if t_1 then t_2 else t_3	conditional
0	constant zero
succ t	successor of t (t + 1)
pred t	predecessor of t (t - 1)
iszero t	if t is zero?

Inductive definition on terms
The **depth** of arithmetic expressions

depth(true)	= 1
depth(false)	= 1
if t_1 then t_2 else t_3	conditional
depth(0)	= 1
succ t_1	successor of t_1 (t_1 + 1)
pred t_1	predecessor of t_1 (t_1 - 1)
iszero t_1	if t_1 is zero?

Inductive definition on terms
The **depth** of arithmetic expressions

depth(true)	= 1
depth(false)	= 1
if t_1 then t_2 else t_3	conditional
depth(0)	= 1
depth(succ t_1)	= 1 + depth(t_1)
depth(pred t_1)	= 1 + depth(t_1)
depth(iszero t_1)	= 1 + depth(t_1)

Inductive definition on terms
The **depth** of arithmetic expressions

depth(true)	$= 1$
depth(false)	$= 1$
depth(if t_1 then t_2 else t_3)	$= 1 + \max(\text{depth}(t_1), \text{depth}(t_2), \text{depth}(t_3))$
depth(0)	$= 1$
depth(succ t_1)	$= 1 + \text{depth}(t_1)$
depth(pred t_1)	$= 1 + \text{depth}(t_1)$
depth(iszero t_1)	$= 1 + \text{depth}(t_1)$

Inductive definition on terms
The **size** of arithmetic expressions

size(true)	$= 1$
size(false)	$= 1$
size(if t_1 then t_2 else t_3)	$= 1 + \text{size}(t_1) + \text{size}(t_2) + \text{size}(t_3)$
size(0)	$= 1$
size(succ t_1)	$= 1 + \text{size}(t_1)$
size(pred t_1)	$= 1 + \text{size}(t_1)$
size(iszero t_1)	$= 1 + \text{size}(t_1)$

Evaluation

Semantics: meaning of programs

t ::=		terms of Boolean expressions
	true	constant true
	false	constant false
	if t_1 then t_2 else t_3	conditional
v ::=		Boolean values
	true	true value
	false	false value

Semantics: meaning of programs

$$\text{if true then } t_2 \text{ else } t_3 \rightarrow t_2 \qquad (\textsc{E-IfTrue})$$

$$\text{if false then } t_2 \text{ else } t_3 \rightarrow t_3 \qquad (\textsc{E-IfElse})$$

$$\frac{t_1 \rightarrow t_1'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \rightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3} \qquad (\textsc{E-If})$$

Semantics: meaning of programs

$$\textbf{if true then } t_2 \textbf{ else } t_3 \to t_2 \qquad (\textsc{E-IfTrue})$$

$$\textbf{if false then } t_2 \textbf{ else } t_3 \to t_3 \qquad (\textsc{E-IfElse})$$

$$\frac{t_1 \to t_1'}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_3 \to \textbf{if } t_1' \textbf{ then } t_2 \textbf{ else } t_3} \qquad (\textsc{E-If})$$

$$\frac{}{\textbf{if true then false else false} \to \textbf{false}}\ \textsc{E-IfTrue}$$

Semantics: meaning of programs

$$\textbf{if true then } \boxed{t_2} \textbf{ else } \boxed{t_3} \to \boxed{t_2} \qquad \text{(E-IfTrue)}$$

$$\textbf{if false then } t_2 \textbf{ else } t_3 \to t_3 \qquad \text{(E-IfElse)}$$

$$\frac{t_1 \to t_1'}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_3 \to \textbf{if } t_1' \textbf{ then } t_2 \textbf{ else } t_3} \qquad \text{(E-If)}$$

$$\frac{}{\textbf{if true then } \boxed{\textbf{false}} \textbf{ else } \boxed{\textbf{false}} \to \boxed{\textbf{false}}} \ \text{E-IfTrue}$$

Semantics: meaning of programs

$$\textbf{if true then } t_2 \textbf{ else } t_3 \to t_2 \qquad \text{(E-IfTrue)}$$

$$\textbf{if false then } t_2 \textbf{ else } t_3 \to t_3 \qquad \text{(E-IfElse)}$$

$$\frac{t_1 \to t_1'}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_3 \to \textbf{if } t_1' \textbf{ then } t_2 \textbf{ else } t_3} \qquad \text{(E-If)}$$

$$\dfrac{\dfrac{}{\textsf{if true then false else false} \to \textsf{false}}\ \text{E-IfTrue}}{\textsf{if (if true then false else false) then true else true} \to \textsf{if false then true else true}}\ \text{E-If}$$

Semantics: meaning of programs

$$\textbf{if true then } t_2 \textbf{ else } t_3 \rightarrow t_2 \qquad \text{(E-IfTrue)}$$

$$\textbf{if false then } t_2 \textbf{ else } t_3 \rightarrow t_3 \qquad \text{(E-IfElse)}$$

$$\frac{t_1 \rightarrow t_1'}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_3 \rightarrow \textbf{if } t_1' \textbf{ then } t_2 \textbf{ else } t_3} \qquad \text{(E-If)}$$

$$\dfrac{\dfrac{}{\textbf{if true then false else false} \rightarrow \textbf{false}}\ \text{E-IfTrue}}{\textbf{if (if true then false else false) then true else true} \rightarrow \textbf{if false then true else true}}\ \text{E-If}$$

On termination of evaluation of Boolean expressions

$$\textbf{if true then } t_2 \textbf{ else } t_3 \rightarrow t_2 \qquad \text{(E-IfTrue)}$$

$$\textbf{if false then } t_2 \textbf{ else } t_3 \rightarrow t_3 \qquad \text{(E-IfElse)}$$

$$\frac{t_1 \rightarrow t_1'}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_3 \rightarrow \textbf{if } t_1' \textbf{ then } t_2 \textbf{ else } t_3} \qquad \text{(E-If)}$$

Each rule translates a larger expression to a smaller one.

$t_1 \rightarrow t_2$ and $\mathrm{size}(t_1) > \mathrm{size}(t_2)$

and also $\mathrm{size}(t) \geqq 0$ for all t.

Semantics: meaning of programs

t ::=	…	terms of Boolean expressions
	0	constant zero
	succ t	successor
	pred t	predecessor
	iszero t	zero test
v ::=	…	Boolean values
	nv	numeric values
nv ::=		numeric values
	0	zero value
	succ nv	successor values

Additional evaluation rules

$$\frac{t_1 \to t_1'}{\textbf{succ } t_1 \to \textbf{succ } t_1'} \quad \text{(E-S{\scriptsize UCC})}$$

$$\textbf{pred } 0 \to 0 \quad \text{(E-P{\scriptsize RED}Z{\scriptsize ERO})}$$

$$\textbf{pred } (\textbf{succ } nv_1) \to t_1 \quad \text{(E-P{\scriptsize RED}S{\scriptsize UCC})}$$

$$\frac{t_1 \to t_1'}{\textbf{pred } t_1 \to \textbf{pred } t_1'} \quad \text{(E-P{\scriptsize RED})}$$

$$\textbf{iszero } 0 \to \textbf{true} \quad \text{(E-I{\scriptsize S}Z{\scriptsize ERO}Z{\scriptsize ERO})}$$

$$\textbf{iszero } (\textbf{succ } nv_1) \to \textbf{false} \quad \text{(E-I{\scriptsize S}Z{\scriptsize ERO}S{\scriptsize UCC})}$$

$$\frac{t_1 \to t_1'}{\textbf{iszero } t_1 \to \textbf{iszero } t_1'} \quad \text{(E-I{\scriptsize S}Z{\scriptsize ERO})}$$

Example: Evaluation of a numerical expression

$$\mathbf{pred}\ (\mathbf{succ}\ (\mathbf{pred}\ 0)) \to \mathbf{pred}\ (\mathbf{succ}\ 0)$$

$$\textbf{pred}\ (\textbf{succ}\ (\underline{\textbf{pred}\ 0})) \to \textbf{pred}\ (\textbf{succ}\ 0)$$

$$\frac{t_1 \to t_1'}{\textbf{succ}\ t_1 \to \textbf{succ}\ t_1'} \qquad \text{(E-Succ)}$$

$$\textbf{pred}\ 0 \to 0 \qquad \text{(E-PredZero)}$$

$$\textbf{pred}\ (\textbf{succ}\ nv_1) \to t_1 \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 \to t_1'}{\textbf{pred}\ t_1 \to \textbf{pred}\ t_1'} \qquad \text{(E-Pred)}$$

$$\textbf{iszero}\ 0 \to \textbf{true} \qquad \text{(E-IsZeroZero)}$$

$$\textbf{iszero}\ (\textbf{succ}\ nv_1) \to \textbf{false} \qquad \text{(E-IsZeroSucc)}$$

$$\frac{t_1 \to t_1'}{\textbf{iszero}\ t_1 \to \textbf{iszero}\ t_1'} \qquad \text{(E-IsZero)}$$

$$\textbf{pred}\ (\textbf{succ}\ (\textbf{pred}\ 0)) \rightarrow \textbf{pred}\ (\textbf{succ}\ 0)$$

$$\frac{}{\textsf{pred}\ 0 \rightarrow}\ \textsc{E-PredZero}$$

$$\frac{t_1 \rightarrow t_1'}{\textbf{succ}\ t_1 \rightarrow \textbf{succ}\ t_1'} \qquad (\textsc{E-Succ})$$

$$\textbf{pred}\ 0 \rightarrow 0 \qquad (\textsc{E-PredZero})$$

$$\textbf{pred}\ (\textbf{succ}\ nv_1) \rightarrow t_1 \qquad (\textsc{E-PredSucc})$$

$$\frac{t_1 \rightarrow t_1'}{\textbf{pred}\ t_1 \rightarrow \textbf{pred}\ t_1'} \qquad (\textsc{E-Pred})$$

$$\textbf{iszero}\ 0 \rightarrow \textbf{true} \qquad (\textsc{E-IsZeroZero})$$

$$\textbf{iszero}\ (\textbf{succ}\ nv_1) \rightarrow \textbf{false} \qquad (\textsc{E-IsZeroSucc})$$

$$\frac{t_1 \rightarrow t_1'}{\textbf{iszero}\ t_1 \rightarrow \textbf{iszero}\ t_1'} \qquad (\textsc{E-IsZero})$$

$$\textbf{pred}\ (\textbf{succ}\ (\textbf{pred}\ 0)) \to \textbf{pred}\ (\textbf{succ}\ 0)$$

$$\frac{}{\text{pred } 0 \to 0} \quad \text{E-PredZero}$$

$$\frac{t_1 \to t_1'}{\textbf{succ}\ t_1 \to \textbf{succ}\ t_1'} \qquad \text{(E-Succ)}$$

$$\textbf{pred}\ 0 \to 0 \qquad \text{(E-PredZero)}$$

$$\textbf{pred}\ (\textbf{succ}\ nv_1) \to t_1 \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 \to t_1'}{\textbf{pred}\ t_1 \to \textbf{pred}\ t_1'} \qquad \text{(E-Pred)}$$

$$\textbf{iszero}\ 0 \to \textbf{true} \qquad \text{(E-IsZeroZero)}$$

$$\textbf{iszero}\ (\textbf{succ}\ nv_1) \to \textbf{false} \qquad \text{(E-IsZeroSucc)}$$

$$\frac{t_1 \to t_1'}{\textbf{iszero}\ t_1 \to \textbf{iszero}\ t_1'} \qquad \text{(E-IsZero)}$$

$$\mathbf{pred}\ (\underline{\mathbf{succ}\ (\mathbf{pred}\ 0)}) \to \mathbf{pred}\ (\mathbf{succ}\ 0)$$

$$\dfrac{\dfrac{}{\mathbf{pred}\ 0 \to 0}\ \text{E-PredZero}}{\mathbf{succ}\ (\mathbf{pred}\ 0) \to}\ \text{E-Succ}$$

$$\frac{t_1 \to t_1'}{\mathbf{succ}\ t_1 \to \mathbf{succ}\ t_1'} \qquad \text{(E-Succ)}$$

$$\mathbf{pred}\ 0 \to 0 \qquad \text{(E-PredZero)}$$

$$\mathbf{pred}\ (\mathbf{succ}\ nv_1) \to t_1 \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 \to t_1'}{\mathbf{pred}\ t_1 \to \mathbf{pred}\ t_1'} \qquad \text{(E-Pred)}$$

$$\mathbf{iszero}\ 0 \to \mathbf{true} \qquad \text{(E-IsZeroZero)}$$

$$\mathbf{iszero}\ (\mathbf{succ}\ nv_1) \to \mathbf{false} \qquad \text{(E-IsZeroSucc)}$$

$$\frac{t_1 \to t_1'}{\mathbf{iszero}\ t_1 \to \mathbf{iszero}\ t_1'} \qquad \text{(E-IsZero)}$$

$$\mathbf{pred}\ (\underline{\mathbf{succ}\ (\mathbf{pred}\ 0)}) \rightarrow \mathbf{pred}\ (\mathbf{succ}\ 0)$$

$$\dfrac{\dfrac{}{\mathbf{pred}\ 0 \rightarrow 0}\ \text{E-PredZero}}{\mathbf{succ}\ (\mathbf{pred}\ 0) \rightarrow \mathbf{succ}\ 0}\ \text{E-Succ}$$

$$\frac{t_1 \rightarrow t_1'}{\mathbf{succ}\ t_1 \rightarrow \mathbf{succ}\ t_1'} \qquad \text{(E-Succ)}$$

$$\mathbf{pred}\ 0 \rightarrow 0 \qquad \text{(E-PredZero)}$$

$$\mathbf{pred}\ (\mathbf{succ}\ nv_1) \rightarrow t_1 \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 \rightarrow t_1'}{\mathbf{pred}\ t_1 \rightarrow \mathbf{pred}\ t_1'} \qquad \text{(E-Pred)}$$

$$\mathbf{iszero}\ 0 \rightarrow \mathbf{true} \qquad \text{(E-IsZeroZero)}$$

$$\mathbf{iszero}\ (\mathbf{succ}\ nv_1) \rightarrow \mathbf{false} \qquad \text{(E-IsZeroSucc)}$$

$$\frac{t_1 \rightarrow t_1'}{\mathbf{iszero}\ t_1 \rightarrow \mathbf{iszero}\ t_1'} \qquad \text{(E-IsZero)}$$

$$\mathbf{pred}\ (\mathbf{succ}\ (\mathbf{pred}\ 0)) \to \mathbf{pred}\ (\mathbf{succ}\ 0)$$

$$\dfrac{\dfrac{\dfrac{}{\mathbf{pred}\ 0 \to 0}\ \text{E-PredZero}}{\mathbf{succ}\ (\mathbf{pred}\ 0) \to \mathbf{succ}\ 0}\ \text{E-Succ}}{\mathbf{pred}\ (\mathbf{succ}\ (\mathbf{pred}\ 0)) \to}\ \text{E-Pred}$$

$$\frac{t_1 \to t_1'}{\mathbf{succ}\ t_1 \to \mathbf{succ}\ t_1'} \qquad \text{(E-Succ)}$$

$$\mathbf{pred}\ 0 \to 0 \qquad \text{(E-PredZero)}$$

$$\mathbf{pred}\ (\mathbf{succ}\ nv_1) \to t_1 \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 \to t_1'}{\mathbf{pred}\ t_1 \to \mathbf{pred}\ t_1'} \qquad \text{(E-Pred)}$$

$$\mathbf{iszero}\ 0 \to \mathbf{true} \qquad \text{(E-IsZeroZero)}$$

$$\mathbf{iszero}\ (\mathbf{succ}\ nv_1) \to \mathbf{false} \qquad \text{(E-IsZeroSucc)}$$

$$\frac{t_1 \to t_1'}{\mathbf{iszero}\ t_1 \to \mathbf{iszero}\ t_1'} \qquad \text{(E-IsZero)}$$

Q: Can we use the E-PredSucc rule instead of E-Pred?

A: No (but why not?)

$$\dfrac{\dfrac{\overline{\textbf{pred } 0 \to 0}\ \textsc{E-PredZero}}{\textbf{succ } (\textbf{pred } 0) \to \textbf{succ } 0}\ \textsc{E-Succ}}{\textbf{pred } (\textbf{succ } (\textbf{pred } 0)) \to \textbf{pred } (\textbf{succ } 0)}\ \textsc{E-Pred}$$

$$\frac{t_1 \to t_1'}{\textbf{succ } t_1 \to \textbf{succ } t_1'} \qquad (\textsc{E-Succ})$$

$$\textbf{pred } 0 \to 0 \qquad (\textsc{E-PredZero})$$

$$\textbf{pred } (\textbf{succ } nv_1) \to t_1 \qquad (\textsc{E-PredSucc})$$

$$\frac{t_1 \to t_1'}{\textbf{pred } t_1 \to \textbf{pred } t_1'} \qquad (\textsc{E-Pred})$$

$$\textbf{iszero } 0 \to \textbf{true} \qquad (\textsc{E-IsZeroZero})$$

$$\textbf{iszero } (\textbf{succ } nv_1) \to \textbf{false} \qquad (\textsc{E-IsZeroSucc})$$

$$\frac{t_1 \to t_1'}{\textbf{iszero } t_1 \to \textbf{iszero } t_1'} \qquad (\textsc{E-IsZero})$$

$$
\dfrac{\dfrac{\overline{\mathbf{pred}\ 0 \to 0}\ \textsc{E-PredZero}}{\mathbf{succ}\ (\mathbf{pred}\ 0) \to \mathbf{succ}\ 0}\ \textsc{E-Succ}}{\mathbf{pred}\ (\mathbf{succ}\ (\mathbf{pred}\ 0)) \to \mathbf{pred}\ (\mathbf{succ}\ 0)}\ \textsc{E-Pred}
$$

$$
\frac{t_1 \to t_1'}{\mathbf{succ}\ t_1 \to \mathbf{succ}\ t_1'} \qquad (\textsc{E-Succ})
$$

$$
\mathbf{pred}\ 0 \to 0 \qquad (\textsc{E-PredZero})
$$

$$
\mathbf{pred}\ (\mathbf{succ}\ nv_1) \to nv_1 \qquad (\textsc{E-PredSucc})
$$

$$
\frac{t_1 \to t_1'}{\mathbf{pred}\ t_1 \to \mathbf{pred}\ t_1'} \qquad (\textsc{E-Pred})
$$

$$
\mathbf{iszero}\ 0 \to \mathbf{true} \qquad (\textsc{E-IsZeroZero})
$$

$$
\mathbf{iszero}\ (\mathbf{succ}\ nv_1) \to \mathbf{false} \qquad (\textsc{E-IsZeroSucc})
$$

$$
\frac{t_1 \to t_1'}{\mathbf{iszero}\ t_1 \to \mathbf{iszero}\ t_1'} \qquad (\textsc{E-IsZero})
$$

E-PredSucc の右辺の t1 は nv1 の誤り.
Q: pred (succ 0) について E-Pred は適用できるか？
A: できない.
理由: succ 0 は値だから．ここまでは，授業のなかでの説明のとおり．値 v については v → v' とならない．これが値の定義．このため Pred の仮定が成立しないため Pred を適用できない.

Q: pred 0 → 0 に0が二箇所に出現するがそれぞれ syntactic element か semantic domain か

. at this

- Previous derivation tree (T_1)

$$\cfrac{\cfrac{\overline{\textbf{pred}\ 0 \to 0}\ \textsc{E-PredZero}}{\textbf{succ}\ (\textbf{pred}\ 0) \to \textbf{succ}\ 0}\ \textsc{E-Succ}}{\textbf{pred}\ (\textbf{succ}\ (\textbf{pred}\ 0)) \to \textbf{pred}\ (\textbf{succ}\ 0)}\ \textsc{E-Pred}$$

- Another derivation tree (T_2)

$$\overline{\textbf{pred}\ (\textbf{succ}\ 0) \to 0}\ \textsc{E-PredSucc}$$

- Their combined meaning is:
  pred(succ (pred 0)) → pred(succ 0) → 0

- pred(succ (pred 0)) $\to^*$ 0

Type System

Types in Computer Science

- A **type system** is a tractable syntactic method for **proving** the absence of certain **program** behaviors by classifying phrases accord to the kinds of values they compute.
    - The notion of types appear in broader field of study: logic, mathematics, and philosophy.

What is a type system?

- A **type system** is a tractable syntactic method for proving the absence of certain program behaviors by **classifying phrases accord to the kinds of values they compute**.
    - a type system can be regarded as calculating a kind of **static approximation to the run-time** behaviors of the term in a program.

What is a type system?

- a **type system** can be regarded as calculating a kind of **static approximation to the run-time** behaviors of the term in a program.

- **Example (Fibonacci series)**

$f_1 = 1,\ f_2 = 2;$
while $f1 < 100$ **do**
  $(f_1, f_2) := (f_2, f_1{+}f_2)$
done

$f_1 = 1,\ f_2 = 2,\ f_3 = 3,\ f_4 = 5,$
$f_5 = 8,\ f_6 = 13,\ f_7 = 21,\ f_8 = 34,$
$f_9 = 55,\ f_{10} = 89, \ldots$

What is a type system?

- A type system is a tractable syntactic method for **proving the absence of certain program behaviors** by classifying phrases accord to the kinds of values they compute.

  function f() =
  a_complex_test_that_always_gives_**true;**

  if f() **then** 5 **else** e^{-2i} (“apple” / π) $\Rightarrow$ ***type error***

What is a type system?

- A type system is a **tractable** syntactic method for proving the absence of certain program behaviors by classifying phrases accord to the kinds of values they compute.
  - Type-checkers are typically built into compilers and linkers and do their job automatically.

Type annotations to assist type-checkers

- f1 = 1, f2 = 2;
  while f1 < 100 **do** (f1, f2) := (f2, f1+f2) **done**

- **int** f1 = 1, f2 = 2;
  while f1 < 100 {
      int t = f1;
      f1 = f2; f2 = t + f2;
  }

Efficiency of type checking algorithms

- We are interested in type checking algorithms that are efficient, but …
  - The computational complexity of **System-F** (the type system of ML) is O(e^*depth of scopes*)
  - For **undecidable type systems**, there are type checking "heuristics" that halt quickly in most cases of practical interest.

What type systems are good for?

- Detecting errors
- Abstraction
- Documentation
- Language safety
- Efficiency

Types in Error Detection

- Programmers working in **richly typed languages** often remark: programs tend to "just work" once they pass the type checker.
- **Example:** Counting money with mixed currency
  - **double** dollar = 5.50, euro = 7.30;
    double sum = **dollar + euro**; // **Bug: missing currency conversion**
  - **type** currency = **Yen**(Int) | **Dollar**(Float) | **Euro**(Float)
    val D2Y = 117.84 and **E2Y** = 146.97
    val *yen* = **Yen**(1000), *dollar* = **Dollar**(5.50), *euro* = **Euro**(7.30);
    function *toYen* **Yen**(*y*) ⇒ Yen(*y*)
    | *toYen* **Dollar**(*d*) ⇒ Yen(floor (**D2Y** * *d*))
    | *toYen* **Euro**(*e*) ⇒ Yen(floor (**E2Y** * *e*));
    var *sum* = **case** toYen(*dollar*), toYen(*euro*): **Yen**(*d*), **Yen**(*e*) → **Yen**(*d+e*)
    var *sum2* = ***dollar + euro*** // Type error detected by the type system

Types in Program Abstraction

- Type system enforces **disciplined programming**
  - Example: Currency conversion library
    type currency = **Yen**(Int) | **Dollar**(Float) | **Euro**(Float)
    function toYen: currency → currency
    function toDollar: currency → currency
    function toEuro: currency → currency
    function total: currency List → currency
- **import** Currency;
  var y = toYen(total [Yen(1000), Dollar(5.50), Euro(7.30)])

- Array module
  - Array.make : int $\rightarrow$ α $\rightarrow$ α array
    let v = Array.make 3 1.0 $\Rightarrow$ [| 1.0; 1.0; 1.0 |]
  - Array.length: α array $\rightarrow$ int
    Array.length v $\Rightarrow$ 3
  - Array.append: α array $\rightarrow$ α array $\rightarrow$ α array
    let v2 = Array.append v v $\Rightarrow$ [| 1.0; 1.0; 1.0; 1.0; 1.0; 1.0 |]
  - Array.fill: α array $\rightarrow$ pos: int $\rightarrow$ len: int $\rightarrow$ α $\rightarrow$ unit
    Array.fill v2 ~pos: 2 ~len: 2 5.0 $\Rightarrow$ [| 1.0; 1.0; 5.0; 5.0; 1.0; 1.0 |]

Types in Language safety

- Safety: a safe language *protects its own abstractions*.
  - **Example (Array)** The programmer expects the content of an array can change only by using the array update operation on the array.
  - int a = 1, b[2];
    printf(“a = %u\n”, a); ⇒ 1
    for (int i = 0; i <= 2; i++) b[i] = 100 - i;
    printf(“a = %u\n”, a); ⇒ 2

Type system vs Safety

	Statically Checked	Dynamically Checked
Safe	ML, Haskell, Java, C#, …	LISP, Basic, Scheme, Python, Ruby, JavaScript, …
Unsafe	C, C++	

Types and Efficiency

C	1.12
Java	1.71
Fortran	1.95
Scala	2.07
LISP*	2.15
Haskell	2.15
C#	2.42
OCaml	3.15
JavaScript*	4.27
Smalltalk*	20.27
Python*	41.79
Ruby*	70.74

Typing

Types (kinds of values)

T ::=		Types
	Bool	type of booleans
	Nat	type of natural numbers

Typing Relation

$$\textbf{true} : \textbf{Bool} \qquad \text{(T-True)}$$

$$\textbf{false} : \textbf{Bool} \qquad \text{(T-False)}$$

$$\frac{t_1 : Bool \qquad t_2 : T \qquad t_3 : T}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_3 : T} \qquad \text{(T-If)}$$

$$0 : \textbf{Nat} \qquad \text{(T-Zero)}$$

$$\frac{t_1 : \textbf{Nat}}{\text{succ } t_1 : \textbf{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{t_1 : \textbf{Nat}}{\text{pred } t_1 : \textbf{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{t_1 : \textbf{Nat}}{\textbf{iszero } t_1 : \textbf{Bool}} \qquad \text{(T-IsZero)}$$

Example: a type of an arithmetic expression

Q: How can we assure ourselves that
“**if iszero** 0 **then** 0 **else** **pred** 0”
evaluates to a natural number?

Q: the type of "if iszero 0 then 0 else pred 0"?

$$\textbf{true} : \textbf{Bool} \qquad \text{(T-True)}$$

$$\textbf{false} : \textbf{Bool} \qquad \text{(T-False)}$$

$$\frac{t_1 : Bool \qquad t_2 : T \qquad t_3 : T}{\textbf{if } t_1 \textbf{ then } t_2 \textbf{ else } t_3 : T} \qquad \text{(T-If)}$$

$$0 : \textbf{Nat} \qquad \text{(T-Zero)}$$

$$\frac{t_1 : \textbf{Nat}}{\text{succ } t_1 : \textbf{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{t_1 : \textbf{Nat}}{\text{pred } t_1 : \textbf{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{t_1 : \textbf{Nat}}{\textbf{iszero } t_1 : \textbf{Bool}} \qquad \text{(T-IsZero)}$$

$$\cfrac{\cfrac{\cfrac{}{0 : \text{Nat}}\ \textsc{T-Zero}}{\text{iszero}\ 0 : \text{Nat}}\ \textsc{T-IsZero} \qquad \cfrac{}{0 : \text{Nat}}\ \textsc{T-Zero} \qquad \cfrac{\cfrac{}{0 : \text{Nat}}\ \textsc{T-Zero}}{\text{pred}\ 0 : \text{Nat}}\ \textsc{T-Pred}}{\text{if iszero}\ 0\ \text{then}\ 0\ \text{else pred}\ 0 : \text{Nat}}\ \textsc{T-If}$$

型推論において iszeo 0 : Nat → iszero 0 : Bool

$$\textbf{true} : \textbf{Bool} \qquad (\textsc{T-True})$$

$$\textbf{false} : \textbf{Bool} \qquad (\textsc{T-False})$$

$$\frac{t_1 : Bool \qquad t_2 : T \qquad t_3 : T}{\textbf{if}\ t_1\ \textbf{then}\ t_2\ \textbf{else}\ t_3 : T} \qquad (\textsc{T-If})$$

$$0 : \textbf{Nat} \qquad (\textsc{T-Zero})$$

$$\frac{t_1 : \textbf{Nat}}{\text{succ}\ t_1 : \textbf{Nat}} \qquad (\textsc{T-Succ})$$

$$\frac{t_1 : \textbf{Nat}}{\text{pred}\ t_1 : \textbf{Nat}} \qquad (\textsc{T-Pred})$$

$$\frac{t_1 : \textbf{Nat}}{\textbf{iszero}\ t_1 : \textbf{Bool}} \qquad (\textsc{T-IsZero})$$

Properties of the type system for arithmetic expressions

- **Uniqueness theorem:**
  If $t : T_1$ and also $t : T_2$ then $T_1 = T_2$.

- **Progress theorem:**
  Suppose t is a well-typed term ($t : T$ for some T). Then either t is a value or else there is some t' with $t \rightarrow t'$.

- **Preservation theorem:**
  If $t : T$ and $t \rightarrow t'$, then $t' : T$.

Properties of the type system for arithmetic expressions

- **Uniqueness theorem:**
  If $t : T_1$ and also $t : T_2$ then $T_1 = T_2$.

- **Progress theorem:**
  Suppose t is a well-typed term ($t : T$ for some T). Then either t is a value or else there is some t' with $t \to t'$.

- **Preservation theorem:**
  If $t : T$ and $t \to t'$, then $t' : T$.

Properties of the type system for arithmetic expressions

- **Uniqueness theorem:**
  If $t : T_1$ and also $t : T_2$ then $T_1 = T_2$.

A well-typed term is not stuck.
An expression is stuck if it is not a value and also it is not irreducible by any evaluation rule.

- **Progress theorem:**
  Suppose t is a well-typed term ($t : T$ for some T). Then either t is a value or else there is some t' with $t \rightarrow t'$.

- **Preservation theorem:**
  If $t : T$ and $t \rightarrow t'$, then $t' : T$.

Properties of the type system for arithmetic expressions

- **Uniqueness theorem:**
  If $t : T_1$ and also $t : T_2$ then $T_1 = T_2$.

- **Progress theorem:**
  Suppose t is a well-typed term ($t : T$ for some T). Then either t is a value or else there is some t' with $t \to t'$.

- **Preservation theorem:**
  If $t : T$ and $t \to t'$, then $t' : T$.

Properties of the type system for arithmetic expressions

- **Uniqueness theorem:**
  If $t : T_1$ and also $t : T_2$ then $T_1 = T_2$.

- **Progress theorem:**
  Suppose t is a well-typed term ($t : T$ for some T). Then either t is a value or else there is some t' with $t \to t'$.

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

- **Preservation theorem:**
  If $t : T$ and $t \to t'$, then $t' : T$.