Type Systems

Authored By Luca Cardelli

ACM Computing Surveys, 1996

Type Systems - Why, What & How? (Informally)

- **Why**: to prevent forbidden(all untrapped and some trapped) errors OR
  "to prove the absence of certain program behaviour"
- **What**: *"tractable syntactic method"*
- **How**: by distinguishing between well typed and ill typed programs, Type Checking OR
  "by classifying phrases according to the kinds of values they compute"

Type Checking and Type System Properties

Type Checking

- No forbidden error $\Longrightarrow$ well behaved $\Longrightarrow$ Strongly Checked
- some untrapped undetected at compilation $\Longrightarrow$ Weakly Checked $\Longrightarrow$ unsafe

Type System Properties

- Decidably Verifiable
- Transparent
- Enforceable
- Prove that "well typed programs are well behaved"

Type Systems for λ-calculus

Syntax for untyped λ-calculus

$M, N :=$		terms
	x	variables
	$\lambda x.M$	functions
	MN	applications

Syntax for first-order F_1 typed λ-calculus

$A, B :=$		types	$M, N :=$		
	K	basic		x	
	$A \to B$	function		$\lambda x : A.M$	
				MN	

Judgments and Rules for F_1

Judgments for F_1

$\Gamma \vdash \diamond$	Γ well-formed environment
	$\Gamma = \{\phi, x_1 : A_1, .., x_n : A_n\}$
$\Gamma \vdash M : A$	M is a well-formed type A in Γ

Type Rules for F_1 (only important)

(Val Fun)

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : \ A \to B}$$

(Val Appl)

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

Let's add Bool Type in F_1

Type Rules for F_1

(Type Bool)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash Bool}$$

(Val True)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash true : Bool}$$

(Val False)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash false : Bool}$$

Val Cond
$$\frac{\Gamma \vdash M : Bool \quad \Gamma \vdash N_1 : A \quad \Gamma \vdash N_2 : A}{\Gamma \vdash (if_A\ M\ then\ N_1\ else\ N_2) : A}$$

Note: if_A is a hint that inferred types for N_1 and N_2 should be compared with A.

Adding Recursive Types in F_1

List Type Example

$List_A \triangleq \mu X.Unit + (A \times X)$

Additional Type Syntax

$A, B :=$	...	types
	$\mu X.A$	Recursive

Additional Operations

unfold$(\mu X.A) = [\mu X.A/X]A$

fold$([\mu X.A/X]A) = \mu X.A$

Type Rules

(Type Rec)

$$\frac{\Gamma, X \vdash A}{\Gamma \vdash \mu X.A}$$

(Val Fold)

$$\frac{\Gamma \vdash M : [\mu X.A/X]A}{\Gamma \vdash fold_{\mu X.A} M : \mu X.A}$$

*iso-recursive approach, unfold(fold(M))=M

Second-order Type System, F_2

Syntax

$A, B :=$	..	types	$M, N :=$	..	terms
	$\forall X.A$	universally quantified		$\lambda X.M$	polymorphic abstraction
				MA	type instantiation

Type Rule

(Val Type Instantiation)

$$\frac{\Gamma \vdash M : \forall X.A \quad \Gamma \vdash B}{\Gamma \vdash MB : [B/X]A}$$

Second-order Type System

Derivation

- $id \triangleq \lambda X.\lambda x : X.x$
- Derive, $M \triangleq id(\forall X.X \to X)(id)$

Subtyping, $F_{1<:}$

An Additional Judgment

- $\Gamma \vdash A <: B$ A is a subtype of B in Γ

Additional Rule

$$\frac{\Gamma \vdash A}{\Gamma \vdash A <: Top}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A <: B}{\Gamma \vdash a : B}$$

$$\frac{\Gamma \vdash A' <: A \quad \Gamma \vdash B <: B'}{\Gamma \vdash A \to B <: A' \to B'}$$

Conclusion

- Highly condensed introduction to Type Systems
- Type Theory, rich and highly expressive but large program are issues