Notes and Excerpts of Type system [1].

Ryan J. Kung
ryankung@ieee.org

March 8, 2019

1 Fundamental

"A **Type System** is to prevent the occurrence of *execution errors* during the running of a program".

1.1 Execution Errors and Safty

There are two kinds of execution errors:

- trapped errors: cause the computation stop immediately.
- untrapped errors: go unnoticed (for a while) and later cause arbitary behavior.

A program fragment is *safe* if it does not cause untrapped errors to occur.

1.2 Expected properties of type systems

- Type systems should be *decidably verifable*
- Type systems should be *transparent*
- Type systems should be *enforceable*

1.3 Formalzation

Once a type system if formlized, we can attempt to prove a *type sondness* theorem stating that *well-typed programms* are *well behaved*.

The first step in formalizing a programming language is to describe its syntax. For most languages of interest this reduces to describing the *syntax of types and terms*.

the next step is to define the *scoping rules of the language*.

When this much is settled, one can proceed to define the type rules of languages. These describe a relation *has-type* of the form $M : A$ between terms

M and types A. Some languages also requie a relation *subtype-of* of the form $A <: B$ between types, and ofen a reation *equal-type* of the form $A = B$ of type equivalence.

static typing evironments are used to record the types of free variables during the procssing of program fragments. The type rules are always formulated with respect to a static environment of the fragment being typechecked. E.g: the has-type relation $M : A$ is associated with a static typing environment Γ that contains information about the free variables of M and A. The relation is wrtten in full as $\Gamma \vdash M : A$.

The final step in formalizing a language is to define its semantics as a relation *has-value* between terms and a collection of *results*.

2 The language of type systems

2.1 Judgements

The description of a type system starts with the the description of a collection of formal utterances called judgement. The empty environment is denoted by $\emptyset$, and the collection of variables $x_1, x_2 ... x_n$ declared in Γ is indicated by $dom(\Gamma)$, the domain of Γ.

A typical judgement has the form: $\Gamma \vdash \mathfrak{I}$ ($\mathfrak{I}$ is an assertion; the free variables of $\mathfrak{I}$ are declared in Γ).

The most important judgement, is the **typing judgement**, which asserts that a term M has a type A with respect to a static typing environment for the free variables of M. It has the form:

$\Gamma \vdash M : A$ M has type A in Γ

Examples

$\emptyset \vdash true : Bool$
true has type *Bool*

$\emptyset, x : Nat \vdash x + 1 : Nat$
$x + 1$ has type *Nat*, provided that x has type *Nat*

$\Gamma \vdash \diamond$
Γ is well-formed

2.2 Type rules

A collection of type rules is called a (formal) type system.

$$\frac{\Gamma_1 \vdash \mathfrak{I}_1 ... \Gamma_n \vdash \mathfrak{I}_n}{\Gamma \vdash \mathfrak{I}}$$

Each type rules is written as a number of *premise* judegements $\Gamma_i \vdash \mathfrak{I}_i$ above a horizonal line, with a single *conclusion* judgement $\Gamma \vdash \mathfrak{I}$ below the time. The process gets off the ground by some intrinsically valid judgment (usually $\emptyset \vdash \diamond$, stating that the empty environment is well formed).

For example:

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash n : Nat} \qquad \frac{\Gamma \vdash M : Nat \ \Gamma \vdash N : Nat}{\Gamma \vdash M + N : Nat}$$

A fundamental rule states that the empty environment is well formed, with no assumptions:

$$(Env\emptyset)$$

$$\frac{}{\emptyset \vdash \diamond}$$

2.3 Type derivations

A derivation in a given type system is a tree of judgments with leaves at the top and a root at the bottom, where each judgment is obtained from the ones immediately above it by some rule of the system. A **valid judgment** is one that can be obtained as the root of a derivation in a given type system.

For Example:

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \diamond}\text{ by (Env }\emptyset)}{\emptyset \vdash 1 : Nat}\text{ by (Val }n) \qquad \dfrac{\dfrac{}{\emptyset \vdash \diamond}\text{ by (Env }\emptyset)}{\emptyset \vdash 2 : Nat}\text{ by (Val }n)}{\emptyset \vdash 1 + 2 : Nat}\text{ by (Val }+)$$

In a given type system, a term M is well typed for an environment Γ, if there is a type A such that $G \vdash M : A$ is a valid judgment; that is, if the term M can be given some type.

The discovery of a derivation (and hence of a type) for a term is called the **type inference problem**.

2.4 First-order Type Systems

2.4.1 pure λ-calculus and typed

The type systems found in most common procedural languages are called **first order**. In type theoretical jargon this means that they lack type parameterization and type abstraction, which are **second order** features.

The purest and most general type system for polymorphism is embodied by a λ-calculus, the minimal fist-order type system can be given by untyped λ-calculus, where the untyped λ-abstraction $\lambda.x.M$ represent a function $\lambda(x) \rightarrow M$(a function of parameter x and result M).

The first-oreder typed λ-calculus is called system F_1. The main change from the untyped λ-calculus is the addition of type annotations for l-abstractions, using the syntax $\lambda x : A.M$, which present $\lambda(x : A) \to M$ (in a typed programming language we would likely include the type of the result, but this is not necessary here.

Syntax of System F_1:

$A, B ::=$	types
$K;\ K \in \mathit{Basic}$	basic types
$A \to B$	function types
$M, N ::=$	terms
x	variable
$\lambda x : A.M$	function
$M\ N$	application

We need only three simple judgments for F_1:

$\Gamma \vdash \diamond$	Γ is a well-formed environment
$\Gamma \vdash A$	A is well-formed type in Γ
$\Gamma \vdash M : A$	M is well-formed term of type A in Γ

The rule (Env $\emptyset$) is the only one that does not require assumptions, and rule (Env x), we must assume that the variable x must not be defined in Γ. when $\Gamma, x : A \to M : B$ has been derived, as in the assumption of (Val Fun), we know that x cannot occur in $dom(\Gamma)$.

Rules of F_1:

(Env Ø) (Env x)

$$\frac{}{\Gamma \to \diamond} \qquad \frac{\Gamma \to A \;\; x \notin dom(\Gamma)}{\Gamma, x : A \vdash \diamond}$$

(Type Const) (Type Arror)

$$\frac{\Gamma \vdash \diamond \;\; K \in Basic}{\Gamma \vdash K} \qquad \frac{\Gamma \vdash A \;\; \Gamma \vdash B}{\Gamma \vdash A \to B}$$

(Val x) (Val Fun) (Val Appl)

$$\frac{\Gamma', x : A, \Gamma'' \vdash \diamond}{\Gamma', x : A, \Gamma'' \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \;\; \Gamma \to N : A}{\Gamma \vdash M \;\; N : B}$$

$(Env\ \emptyset)$

$$\frac{}{\Gamma \to \diamond}$$

$(Env\ x)$

$$\frac{\Gamma \to A\ x \notin dom(\Gamma)}{\Gamma, x : A \vdash \diamond}$$

$(Type\ Const)$ $(Type\ Arror)$

$$\frac{\Gamma \vdash \diamond\ K \in Basic}{\Gamma \vdash K} \qquad \frac{\Gamma \vdash A\ \Gamma \vdash B}{\Gamma \vdash A \to B}$$

$(Val\ x)$

$$\frac{\Gamma', x : A, \Gamma'' \vdash \diamond}{\Gamma', x : A, \Gamma'' \vdash x : A}$$

$(Val\ Fun)$

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

$(Val\ Appl)$

$$\frac{\Gamma \vdash M : A \to B\ \Gamma \to N : A}{\Gamma \vdash M\ \ N : B} \tag{1}$$

The rule (Type const) and (Type Arrow) construct types. the rule (Val x)

extras an assumption from an environment: We use the notaion $\Gamma', x : A, \Gamma''$ to indicate that $x : A$ occurs somewhere in the environment.

Now that we have examined the basic structure of a simple first-order type system, we can begin enriching it to bring it closer to the type structure of actual programming languages. We are going to add a set of rules for each new type construction, following a fairly regular pattern. We begin with some basic data types: the type Unit.

Union Type:

$$\frac{\overset{(TypeUnit)}{\Gamma \vdash \diamond}}{\Gamma \vdash Unit} \qquad \frac{\overset{(ValUnit)}{\Gamma \vdash \diamond}}{\Gamma \vdash unit : Unit} \tag{2}$$

Bool Type:

$$\begin{array}{ll} \frac{\overset{(TypeBool)}{\Gamma \vdash \diamond}}{\Gamma \vdash Bool} & \frac{\overset{(ValTrue)}{\Gamma \vdash \diamond}}{\Gamma \vdash true : Bool} \frac{\overset{(ValFalse)}{\Gamma \vdash \diamond}}{\Gamma \vdash false : Bool} \\ \frac{\overset{(Val\ Cond)}{\Gamma \vdash M : Bool\ \Gamma \vdash N_1 : A\ \Gamma \vdash N_2 : A}}{\Gamma \vdash (if_A\ M\ then N_1\ else\ N_2 : A)} & \end{array} \tag{3}$$

Nat Type:

$$\begin{array}{ll} \frac{\overset{(TypeNat)}{\Gamma \vdash \diamond}}{\Gamma \vdash Nat} & \frac{\overset{(ValZero)}{\Gamma \vdash \diamond}}{\Gamma \vdash 0 : Nat} \frac{\overset{(ValSucc)}{\Gamma \vdash M : Nat}}{\Gamma \vdash succ\ M : Nat} \\ \frac{\overset{(Val\ Cond)}{\Gamma \vdash M : Nat}}{\Gamma \vdash pred\ M : Nat} & \frac{\overset{(var\ IsZero)}{\Gamma \vdash M : Nat}}{\Gamma \vdash isZero\ M : Bool} \end{array} \tag{4}$$

References

[1] Luca Cardelli. Type systems. *ACM Comput. Surv.*, 28(1):263–264, March 1996.