Lecture 14: Type Checking

Carsten Schürmann

October 10, 2001

Some aspects of Tiger's semantics can be captured by a type system which we discuss in this and the next lecture. In the previous lecture, we have introduced a mathematical way to describe formal systems in general which led to the definition of a symbol table. The purpose of these tables is to represent binding between symbols and names and other entities, such as types or values. In the context of type-checking we consider symbols that map variables names to types.

Specifically we have introduced the notion of a judgment that expresses if something is true, such as "expression e has type τ in context Γ" or "expression e evaluates to value v". To be more space efficient, we abbreviate the first judgment by $\Gamma \vdash e : \tau$.

What counts as evidence of the truth of a judgment? In general, we use inference rules with premisses and conclusions to build derivations which we take as evidence for judgments. The name of a derivation of judgment J is denoted by the calligraphic letter $\mathcal{D}$ above J written as

$$\begin{array}{c}\mathcal{D} \\ J\end{array}\ .$$

1 Syntactical Categories for Tiger

We begin our excursion into the world of type checking for Tiger with a formal treatment of two things. First, we formalize the abstract syntax of Tiger as follows. Note that for this lecture we do not consider arrays and records, and omit therefore all subsequent constructions. To add them will be the goal of the next lecture.

$$\begin{array}{llll}
\text{Declarations:} & d & ::= & \cdot \mid f(x_1 : \tau_1 \ldots x_n : \tau_n) = e, d \\
 & & & \mid f(x_1 : \tau_1 \ldots x_n : \tau_n) : \tau = e, d \\
\text{Expressions:} & e & ::= & () \mid e_1 ; e_2 \mid \overline{n} \mid \overline{s} \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \mid e_1 / e_2 \\
 & & & \mid e_1 = e_2 \mid e_1 \neq e_2 \mid e_1 > e_2 \mid e_1 < e_2 \mid e_1 \geq e_2 \mid e_1 \leq e_2 \\
 & & & \mid e_1 \& e_2 \mid e_1 | e_2 \\
 & & & \mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \mid \text{if } e_1 \text{ then } e_2 \mid \text{while } e_1 \text{ do } e_2 \\
 & & & \mid \text{for } x = e_1 \text{ to } e_2 \text{ do } e_3 \mid \text{break} \mid \text{let } d \text{ in } e \text{ end}
\end{array}$$

Second we characterize types, that represent the meaning of all objects of one type.

$$\text{Types:} \quad \tau \quad ::= \quad \text{int} \mid \text{string} \mid \text{unit} \mid \tau_1 \to \tau_2$$

The type int, for example, is the type of all integer objects, and we will use this type in order to decide if two expressions can be added, subtracted, or compared. The type string captures the meaning of something being a string, the type unit simply stands for the unit (or void) type.

When programming in Tiger one function after the other is being declared, and must be stored some place for future reference. For the purpose of type checking however, it is less important how a function is implemented, it is more important, what its type is. Otherwise references to this functions that occur later in the code cannot be type-checked. This information is summarized in contexts. A context is a list of function symbols stored together with their types, which may shadow previous declarations.

$$\text{Contexts:} \quad \Gamma \quad ::= \quad \cdot \mid \Gamma, f : \tau$$

Mathematically, we write $\Gamma(f) = \tau$ for looking up the type of f in Γ. Finally we have to define one more syntactic category, which we call flag for the lack of a better word. Tiger programs may contain so called break expressions which cause the operational semantics to interrupt the current computation and leave the current body of a while or a for statement. Naturally, in order to judge if a break statement is well-placed in a program or not, one must ensure that it occurs only within a while or for block. This information is captured by flag ι.

$$\text{Flag:} \quad \iota \quad ::= \quad 0 \mid 1$$

If $\iota = 0$ then break statements are forbidden, and if $\iota = 1$ then they are legal.

2 Typing Judgment and inference rules

The two judgments which we will be defining in this section are

$$\begin{array}{rl} \text{a typing judgment for expressions:} & \Gamma \vdash^{\iota} e : \tau \\ \text{and a typing judgment for declarations:} & \Gamma \vdash^{\iota} d/e : \tau. \end{array}$$

The first characterizes valid Tiger expressions e. In its definition, ι keeps track of if we are inside a while or for block, or not. Γ contains all typing information of functions already declared, and τ is the type of the expression.

The form of the second judgment has a slightly more complicated form, and it seems to assign types to a list of declarations d and an expression e simultaneously. This however is not the case. On the contrary, the judgment should be read as a *residual judgment*. If it is possible to find a derivation $\mathcal{D}$ of

$$\begin{array}{c} \mathcal{D} \\ \Gamma \vdash^{\iota} d/e : \tau \end{array}$$

then e has type τ after executing all declarations in d and adding the respective declarations to Γ. Consequently, a closed tiger program (which consists only of a list of declarations d) is well-typed if and only if a derivation of $\mathcal{D}$ of judgment

$$\begin{array}{c}\mathcal{D}\\ \cdot \vdash^0 d/() : \text{unit}\end{array}$$

exists. In the remainder of this section, we define the meaning of those two judgments in terms of inference rules. In particular we define what it means for any Tiger construct to be well-typed in terms of its components. This presentation of typing rules is not included in Appel's book. In this presentation, we follow however closely the layout of the description of Tiger in the appendix of Appel's book.

2.1 Declarations

$$\frac{\Gamma \vdash^\iota e : \tau}{\Gamma \vdash^\iota \cdot/e : \tau}\ \textsf{emtpy}$$

$$\frac{\begin{array}{l}\Gamma, f : \tau_1 \to \ldots \to \tau_n \to \text{unit}, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash^\iota e : \text{unit}\\ \Gamma, f : \tau_1 \to \ldots \to \tau_n \to \text{unit} \vdash^\iota d/e' : \tau'\end{array}}{\Gamma \vdash^\iota f(x_1 : \tau_1 \ldots x_n : \tau_n) = e, d/e' : \tau'}\ \textsf{fundec1}$$

$$\frac{\begin{array}{l}\Gamma, f : \tau_1 \to \ldots \to \tau_n \to \tau, x_1 : \tau_1, \ldots, x_n : \tau_n \vdash^\iota e : \tau\\ \Gamma, f : \tau_1 \to \ldots \to \tau_n \to \tau \vdash^\iota d/e' : \tau'\end{array}}{\Gamma \vdash^\iota f(x_1 : \tau_1 \ldots x_n : \tau_n) : \tau = e, d/e' : \tau'}\ \textsf{fundec2}$$

2.2 Valueless expression

$$\frac{}{\Gamma \vdash^\iota () : \text{unit}}\ \textsf{unit}$$

2.3 Sequencing

$$\frac{\Gamma \vdash^\iota e_1 : \tau' \qquad \Gamma \vdash^\iota e_2 : \tau}{\Gamma \vdash^\iota e_1; e_2 : \tau}\ \textsf{seq}$$

2.4 Integer literal

$$\frac{}{\Gamma \vdash^\iota \overline{n} : \text{int}}\ \textsf{nat}\quad \text{where } n \text{ is an integer number}$$

2.5 String literal

$$\frac{}{\Gamma \vdash^{\iota} \overline{s} : \text{string}}\,\textsf{nat} \quad \text{where } s \text{ is a character string}$$

2.6 Negation

$$\frac{\Gamma \vdash^{\iota} e : \text{int}}{\Gamma \vdash^{\iota} -e : \text{int}}\,\textsf{neg}$$

2.7 Function Call

$$\frac{\begin{array}{l}\Gamma(f) = \tau_1 \to \ldots \to \tau_n \to \tau \\ \Gamma \vdash^{\iota} e_1 : \tau_1 \\ \vdots \\ \Gamma \vdash^{\iota} e_n : \tau_n\end{array}}{\Gamma \vdash^{\iota} f(e_1, \ldots, e_n) : \tau}\,\textsf{app}$$

2.8 Arithmetic

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 + e_2 : \text{int}}\,\textsf{plus} \qquad \frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 * e_2 : \text{int}}\,\textsf{mult}$$

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 - e_2 : \text{int}}\,\textsf{minus} \qquad \frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 / e_2 : \text{int}}\,\textsf{div}$$

2.9 Comparison

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 = e_2 : \text{int}}\,\textsf{eq} \qquad \frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 \neq e_2 : \text{int}}\,\textsf{neq}$$

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 > e_2 : \text{int}}\,\textsf{gt} \qquad \frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 < e_2 : \text{int}}\,\textsf{lt}$$

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 \geq e_2 : \text{int}}\,\textsf{geq} \qquad \frac{\Gamma \vdash^{\iota} e_1 : \text{int} \qquad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 \leq e_2 : \text{int}}\,\textsf{leq}$$

2.10 String Comparison

$$\frac{\Gamma \vdash^{\iota} s_1 : \text{string} \quad \Gamma \vdash^{\iota} s_2 : \text{string}}{\Gamma \vdash^{\iota} s_1 = s_2 : \text{string}}\,\textsf{eq} \qquad \frac{\Gamma \vdash^{\iota} s_1 : \text{string} \quad \Gamma \vdash^{\iota} s_2 : \text{string}}{\Gamma \vdash^{\iota} s_1 \neq s_2 : \text{string}}\,\textsf{neq}$$

$$\frac{\Gamma \vdash^{\iota} s_1 : \text{string} \quad \Gamma \vdash^{\iota} s_2 : \text{string}}{\Gamma \vdash^{\iota} s_1 > s_2 : \text{string}}\,\textsf{gt} \qquad \frac{\Gamma \vdash^{\iota} s_1 : \text{string} \quad \Gamma \vdash^{\iota} s_2 : \text{string}}{\Gamma \vdash^{\iota} s_1 < s_2 : \text{string}}\,\textsf{lt}$$

$$\frac{\Gamma \vdash^{\iota} s_1 : \text{string} \quad \Gamma \vdash^{\iota} s_2 : \text{string}}{\Gamma \vdash^{\iota} s_1 \geq s_2 : \text{string}}\,\textsf{geq} \qquad \frac{\Gamma \vdash^{\iota} s_1 : \text{string} \quad \Gamma \vdash^{\iota} s_2 : \text{string}}{\Gamma \vdash^{\iota} s_1 \leq s_2 : \text{string}}\,\textsf{leq}$$

2.11 Boolean Operators

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \quad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 \& e_2 : \text{int}}\,\textsf{and} \qquad \frac{\Gamma \vdash^{\iota} e_1 : \text{int} \quad \Gamma \vdash^{\iota} e_2 : \text{int}}{\Gamma \vdash^{\iota} e_1 | e_2 : \text{int}}\,\textsf{or}$$

2.12 If-then-else

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \quad \Gamma \vdash^{\iota} e_2 : \tau \quad \Gamma \vdash^{\iota} e_3 : \tau}{\Gamma \vdash^{\iota} \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau}\,\textsf{ifthenelse}$$

2.13 If-then

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \quad \Gamma \vdash^{\iota} e_2 : \text{unit}}{\Gamma \vdash^{\iota} \text{if } e_1 \text{ then } e_2 : \text{unit}}\,\textsf{ifthen}$$

2.14 While

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \quad \Gamma \vdash^{1} e_2 : \text{unit}}{\Gamma \vdash^{\iota} \text{while } e_1 \text{ do } e_2 : \text{unit}}\,\textsf{while}$$

2.15 For

$$\frac{\Gamma \vdash^{\iota} e_1 : \text{int} \quad \Gamma \vdash^{\iota} e_2 : \text{int} \quad \Gamma \vdash^{1} e_3 : \text{unit}}{\Gamma \vdash^{\iota} \text{for } x = e_1 \text{ to } e_2 \text{ do } e_3 : \text{unit}}\,\textsf{for}$$

2.16 Break

$$\frac{}{\Gamma \vdash^{1} \text{break} : \text{unit}}\,\textsf{break}$$

2.17 Let

$$\frac{\Gamma \vdash^{\iota} d/e : \tau}{\Gamma \vdash^{\iota} \text{let } d \text{ in } e \text{ end} : \tau}\,\textsf{let}$$

In the next lecture, we extend this design to accommodate arrays records, assignment and l-values.