Geometry Aware Types For Reference Frames (Gator)

Dietrich Geisler

1 Syntax

Literals in our language can be scalar numbers, vectors, or matrices:

$$
\begin{aligned}
s &\in \mathbb{R} \\
v_n &::= [s_1, s_2, \ldots, s_n] \\
m_{n_1 \times n_2} &::= [[s_{11}, s_{12}, \ldots, s_{1n_2}], \ldots, [s_{n_1 1}, s_{n_1 2}, \ldots, s_{n_1 n_2}]]
\end{aligned}
$$

Expressions can be variables, literals, or unary/binary operations:

$$
\begin{aligned}
&x \in \text{variables} \\
&c ::= s \mid v_n \mid m_{n_1 \times n_2} \\
&e ::= c \mid \tau\ x = e \mid x = e \mid -e \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid
\end{aligned}
$$

The foundation of our type system is a *geometric type*. Our type system consists of these geometric types, function types, and built-in types:

$$
\begin{aligned}
&n \in \mathbb{N} &\qquad& \nu ::= \top_n \mid \bot_n \mid T \\
&T \in \text{tags} && \tau ::= \mathsf{unit} \mid \mathsf{scalar} \mid \nu \mid \nu_1 \to \nu_2
\end{aligned}
$$

Function types $\nu_1 \to \nu_2$ are not ordinary functions; they are matrices used to map from one vector space to another using matrix–vector multiplication. As a convenience (as well as to match expected GLSL behavior), the top type $\top_n$ can be written in code as vec_n and the $\top_{n_1} \to \top_{n_2}$ type can be written in code as $\mathsf{mat}n_1\mathsf{x}n_2$. The $\bot_n$ type is purely meant for use by the typechecker and so is not part of the surface syntax of the language.

2 Subtype Ordering

We define a judgment $\tau_1 \leq \tau_2$ for subtyping on ordinary types. Subtyping is reflexive and transitive, as usual:

$$
\frac{}{\tau \leq \tau} \qquad\qquad \frac{\tau_1 \leq \tau_2 \qquad \tau_2 \leq \tau_3}{\tau_1 \leq \tau_3}
$$

The null type unit acts as a top type for all of τ:

$$\frac{}{\tau \leq \mathsf{unit}}$$

We also introduce a partial order on tags ν, of the form $\Delta \vdash \nu_1 \leq_\Delta \nu_2$. This ordering refers to a context Δ, which is a map from tags T to matrix types ν:

$$\frac{\Delta \vdash \Delta(\nu_1) = \nu_2}{\Delta \vdash \nu_1 \leq_\Delta \nu_2} \quad \Delta(T) = \nu$$

This relation has the usual reflexivity and transitivity properties:

$$\frac{}{\Delta \vdash \nu \leq_\Delta \nu} \qquad\qquad \frac{\Delta \vdash \nu_1 \leq_\Delta \nu_2 \qquad \Delta \vdash \nu_2 \leq_\Delta \nu_3}{\Delta \vdash \nu_1 \leq_\Delta \nu_3}$$

Members of ν, namely vectors and tagged vectors, have a top type representing general vectors of dimension n as $\top_n$. Additionally, we also introduce a bottom type $\bot_n$ for each vector dimension. We naturally have the simple rule

$$\frac{}{\Delta \vdash \bot_n \leq_\Delta \top_n}$$

The map Δ should not put anything above the top types $\top_n$ and the bottom types $\bot_n$. In particular, we require the rules

$$\Delta(\nu) \neq \bot_n \qquad \text{or} \qquad \Delta(\top_n) \neq \nu$$

Additionally, we need a rule to subsume the bottom type into values correctly since Δ only 'points' towards the top type. In particular, we require the rule:

$$\frac{\Delta \vdash \nu \leq_\Delta \top_n}{\Delta \vdash \bot_n \leq_\Delta \nu}$$

This rule, along with the constructions previously described, gives the standard properties of a top and bottom. In particular, for any ν, Δ, there exists a $\top_n$ and $\bot_n$ such that

$$\Delta \vdash \bot_n \leq_\Delta \nu \quad \text{and} \quad \Delta \vdash \nu \leq_\Delta \top_n$$

The function-like transformation matrix types, $\nu_1 \to \nu_2$, have the same standard subtyping relationship as ordinary function types:

$$\frac{\Delta \vdash \nu_1' \leq_\Delta \nu_1 \qquad \Delta \vdash \nu_2 \leq_\Delta \nu_2'}{\Delta \vdash \nu_1 \to \nu_2 \leq \nu_1' \to \nu_2'}$$

As with vectors, we can define matrix functions have a $\top_{n_1 \to n_2}$ and $\bot_{n_1 \to n_2}$ for any pair of dimensions n_1, n_2. These constructions are just maps $\nu_1 \to \nu_2$, namely:

$$\top_{n_1 \to n_2} = \bot_{n_1} \to \top_{n_2} \qquad\qquad \bot_{n_1 \to n_2} = \top_{n_1} \to \bot_{n_2}$$

As with vectors, these rules produce the usual property of lattices, namely that every map is 'between' a given pair top and bottom values. In particular, for any ν_1, ν_2, Δ, there exists a $\top_{n_1 \to n_2}$ and $\bot_{n_1 \to n_2}$ such that:

$$\Delta \vdash \bot_{n_1 \to n_2} \leq \nu_1 \to \nu_2 \quad \text{and} \quad \Delta \vdash \nu_1 \to \nu_2 \leq \top_{n_1 \to n_2}$$

Additionally, for any ν_1, ν_2, Δ, there is no $\top_{n_1 \to n_2}$ or $\bot_{n_1 \to n_2}$ such that

$$\Delta \vdash \top_{n_1 \to n_2} \leq \nu_1 \to \nu_2 \quad \text{or} \quad \Delta \vdash \nu_1 \to \nu_2 \leq \bot_{n_1 \to n_2}$$

3 Static Semantics

This typing judgement is a map from an expression to the type of that expression under some variable context Γ, from variable names to types, and some tag context Δ defined above.

3.1 Subsumption

Any type in a given context can be cast "up" at any time.

$$\frac{\tau_1 \leq \tau_2 \qquad \Gamma, \Delta \vdash e : \tau_1, \Gamma}{\Gamma, \Delta \vdash e : \tau_2, \Gamma} \qquad \frac{\Delta \vdash \nu_1 \leq_\Delta \nu_2 \qquad \Gamma, \Delta \vdash e : \nu_1, \Gamma}{\Gamma, \Delta \vdash e : \nu_2, \Gamma}$$

3.2 Constants and Variable Declarations

Declaring constants produce the types one would expect:

$$\frac{}{\Gamma, \Delta \vdash () : \mathsf{unit}, \Gamma} \qquad \frac{}{\Gamma, \Delta \vdash s : \mathsf{scalar}, \Gamma}$$

Vector and matrix literals take on their respective bottom types of the appropriate dimension (note the swap in dimensions for the bottom type as a result of treating vectors as column vectors).

$$\frac{}{\Gamma, \Delta \vdash v_n : \bot_n, \Gamma} \qquad \frac{}{\Gamma, \Delta \vdash m_{n_1 \times n_2} : \bot_{n_2 \times n_1}, \Gamma}$$

Variables can be declared and assigned. Assignment is required at declaration time. Note that the entire variable must be reassigned in the case of vectors and matrices:

$$\frac{\Gamma, \Delta \vdash e : \tau, \Gamma'}{\Gamma, \Delta \vdash \tau\ x := e : \mathsf{unit}, \Gamma', x \mapsto \tau} \qquad \frac{\Gamma, \Delta \vdash \Gamma(x) : \tau \qquad \Gamma, \Delta \vdash e : \tau, \Gamma'}{\Gamma, \Delta \vdash x := e : \mathsf{unit}, \Gamma'}$$

3.3 Binary Operations

All operators on scalars work as one might expect.

Types are closed under addition and scalar multiplication

$$\frac{\Gamma, \Delta \vdash e_1 : \tau, \Gamma' \qquad \Gamma', \Delta \vdash e_2 : \tau, \Gamma''}{\Gamma, \Delta \vdash e_1 + e_2 : \tau, \Gamma''} \quad \tau \neq \mathsf{unit}$$

$$\frac{\Gamma, \Delta \vdash e_1 : \tau, \Gamma' \qquad \Gamma', \Delta \vdash e_2 : \mathsf{scalar}, \Gamma''}{\Gamma, \Delta \vdash e_1 * e_2 : \tau, \Gamma'}$$

Component multiplication can be defined as a mathematical operation, but makes little formal sense. Such operations are allowed, but don't interact with spaces and tags and result in a complete lack of information about a resulting matrix.

$$\frac{\Gamma, \Delta \vdash e_1 : \top_n, \Gamma' \qquad \Gamma', \Delta \vdash e_2 : \top_n, \Gamma''}{\Gamma, \Delta \vdash e_1 \mathbin{.*} e_2 : \top_n, \Gamma''}$$

$$\frac{\Gamma, \Delta \vdash e_1 : \top_{n_1 \to n_2}, \Gamma' \qquad \Gamma', \Delta \vdash e_2 : \top_{n_1 \to n_2}, \Gamma''}{\Gamma, \Delta \vdash e_1 \mathbin{.*} e_2 : \top_{n_1 \to n_2}, \Gamma''}$$

Matrix multiplication is both a way of transforming from one tag to another and for composing two matrix functions together. Note that these operations are *not* commutative and vectors are treated as column vectors with regards to matching dimensions.

$$\frac{\Gamma, \Delta \vdash e_1 : \nu_1 \to \nu_2, \Gamma' \qquad \Gamma', \Delta \vdash e_2 : \nu_1, \Gamma''}{\Gamma, \Delta \vdash e_1 * e_2 : \nu_2, \Gamma''}$$

$$\frac{\Gamma, \Delta \vdash e_1 : \nu_2 \to \nu_3, \Gamma' \qquad \Gamma', \Delta \vdash e_2 : \nu_1 \to \nu_2, \Gamma''}{\Gamma, \Delta \vdash \ e_1 * e_2 : \nu_1 \to \nu_3, \Gamma''}$$

4 Dynamic Semantics

The operational semantics of this language map, in a single step, from a command to a constant and a state σ. σ itself is, as usual, a map from variable names to constants.

4.1 Substitution

Substitutions work as expected on the environment σ.

$$\frac{}{\tau\ x := c, \sigma \to (), \sigma[c/x]} \qquad \frac{e, \sigma \to e', \sigma}{\tau\ x := e, \sigma \to \tau\ x := e', \sigma}$$

4.2 Mathematical Operations

As usual, we can reduce on either side of the mathematical operations $\odot \in \{+, *, .*\}$

$$\frac{e_1, \sigma \to e_1', \sigma}{e_1 \odot e_2, \sigma \to e_1' \odot e_2, \sigma} \qquad \frac{e_2, \sigma \to e_2', \sigma}{c \odot e_2, \sigma \to c \odot e_2', \sigma}$$

Addition and scalar multiplication have standard mathematical meaning. Similarly, vector and matrix multiplication have standard mathematical meaning. Component-wise multiplication is identical to matrix addition, except using multiplication of numbers in place of addition of numbers. For readability, formal semantics of each of these operations will be ommitted.