Sistema de Tipos para CPP
Curso de Lenguajes de Programación — Universidad ORT Uruguay
2020

0. Introducción. En este documento presentamos un sistema de tipos para el lenguaje CPP, inspirado en las especificaciones del texto de A. Ranta[1]. Algunos detalles han sido provistos en este documento que no están explícitos en la referencia precedente, y otros han sido modificados.

Se comienza exponiendo una sintaxis abstracta del lenguaje. Luego se explican las cinco formas de juicio utilizadas en el sistema de tipos y, finalmente, se dan las reglas, en el formato usual:

$$\frac{P_1 \; P_2 \; \ldots \; P_n}{C}$$

que indica que el conocimiento de las *premisas* $P_1 \; P_2 \; \ldots \; P_n$ permite inferir la *conclusión* C. Premisas y conclusiones son siempre casos de las varias formas de juicio que se explican en la sección 2. En varios casos las reglas deben ser complementadas por condiciones laterales adicionales, que se expresan a la derecha de la raya de inferencia.

Leyendo las reglas de abajo hacia arriba se tiene información de la forma:

> Para verificar C, alcanza con verificar $P_1 \; P_2 \; \ldots \; P_n$,

la cual aproxima el sistema a un algoritmo de *type-checking*. De este modo, el presente documento puede verse como una especificación y plano básico para el diseño de un *type-checker* para CPP.

1. Sintaxis abstracta.

p	::=	$\overline{\delta}$	(Un programa es una lista de definiciones.)
δ	::=	fun $\rho \; f \; \overline{x : \alpha} \; \overline{s}$	
s	::=	e	expresión
	|	$\alpha \; \overline{x}$	declaración de variables
	|	$\alpha \; x = e$	inicialización de una variable
	|	return	vacío
	|	return e	valor
	|	$\{\overline{s}\}$	bloque
	|	while $e \; s$	
	|	if $e \; s \; s$	
α	::=	bool | int | double | string	
ρ	::=	α | void	

[1] www1.digitalgrammars.com/ipl-book/assignments/assignment2/assignment2.html

$$
\begin{array}{rll}
e & ::= & \mathsf{true} \mid \mathsf{false} \mid \mathsf{Int}\, n \mid \mathsf{Double}\, d \mid \mathsf{String}\, w \quad \text{literales} \\
& \mid & x \quad \text{variable} \\
& \mid & f(\overline{e}) \quad \text{invocación de función} \\
& \mid & x\texttt{++} \mid x\texttt{--} \mid \texttt{++}x \mid \texttt{--}x \\
& \mid & e\texttt{+}e \mid e\texttt{-}e \mid e\texttt{*}e \mid e\texttt{/}e \\
& \mid & e\texttt{<}e \mid e\texttt{<=}e \mid e\texttt{==}e \mid e\texttt{!=}e \mid e\texttt{>=}e \mid e\texttt{>}e \\
& \mid & e\texttt{\&\&}e \mid e \mid\mid e \\
& \mid & x\texttt{=}e
\end{array}
$$

x, f son nombres (identificadores).
n es un entero, d un valor de punto flotante, y w un string.

2. Juicios del sistema de tipos. Se utilizan cinco formas de juicio:

- p valid —significa que el programa p es válido en cuanto a uso de tipos y declaraciones.
- $p \triangleright \Sigma$ —significa que el programa p tiene asociada o genera una *signatura de funciones* Σ. Una signatura de funciones es una tabla $f \mapsto (\overline{\alpha}, \rho)$, es decir, que asocia a cada nombre de función definida en un programa, una pareja formada por la lista de tipos correspondientes a los argumentos y el tipo de retorno de la función. En una signatura, un nombre de función no puede aparecer más de una vez, reflejando que, en CPP, una función no puede ser redefinida dentro del mismo programa. Una signatura se usa para controlar que el programa correspondiente realiza un uso correcto de las funciones definidas en él.
- $\Sigma \vdash p$ ok —significa que, bajo la signatura de funciones Σ, el programa p es consistente en cuanto a uso de tipos y declaraciones.
- $\Sigma; \Gamma \vdash_\rho \overline{s}$ ok —significa que la secuencia de instrucciones (*statements*) $\overline{s}$ es válida en cuanto a uso de tipos y declaraciones, bajo la signatura de funciones Σ y el *contexto de declaraciones de variables* Γ, siendo además ρ el tipo de retorno de la función donde esas instrucciones se encuentran. Un contexto de declaraciones de variables es un *stack de tablas*, donde cada tabla corresponde a un *ámbito* (*scope*) de variables y está formada por asociaciones $x \mapsto \alpha$ de un nombre de variable a un tipo. En cada tabla un nombre de variable aparece a lo sumo una vez, reflejando que, en un ámbito dado, no es posible redeclarar variables. Los ámbitos pueden, sin embargo, anidarse (mediante la introducción de *bloques* de instrucciones) y es válido que cualquier variable se redeclare en un ámbito (bloque) interior a otro en el que la variable ya hubiera sido declarada. Esta característica es la que motiva la utilización de una estructura de stack para el contexto de declaraciones de variables. La pareja formada por una signatura de funciones y un contexto de declaraciones de variables es a menudo llamada un *ambiente* y también una *tabla de símbolos*. Finalmente, el tipo ρ de retorno de la función que contiene la secuencia de instrucciones que está siendo verificada se utiliza para asegurar la validez de tipo de las instrucciones `return`.

- $\Sigma; \Gamma \vdash e : \alpha$ —significa que la expresión e tiene tipo α bajo la signatura de funciones Σ y el contexto de declaraciones de variables Γ.

3. Reglas del sistema de tipos.

Validez de programa.

$$\frac{p \triangleright \Sigma \qquad \Sigma \vdash p \ \mathsf{ok}}{p \ \mathsf{valid}}$$

(Un programa es válido si genera una signatura de funciones Σ y luego resulta consistente bajo esa signatura.)
(Cada premisa de la regla precedente corresponde a una pasada del type-checker sobre el programa: primero se genera la signatura y luego se chequea el programa completo bajo la signatura generada. En la primera pasada sólo se analizan las firmas de las funciones que componen el programa; en la segunda, se analiza el cuerpo de código de cada función. Por supuesto, el type checker puede terminar en error durante cualquiera de las dos pasadas.)

Generación de signatura.

$$\frac{}{. \triangleright .}$$

$$\frac{p \triangleright \Sigma}{p \ \ (\mathsf{fun}\ \rho\ f\ \overline{x : \alpha}\ \overline{s}) \quad \triangleright \quad \Sigma, f \mapsto (\overline{\alpha}, \rho)} \begin{cases} f \text{ no declarada en } \Sigma \\ \overline{x} \text{ sin repeticiones} \end{cases}$$

(Estas reglas implementan la generación de la signatura de funciones de un programa por simple recursión sobre éste. Debe recordarse que un programa es una lista de definiciones de funciones. La primera regla corresponde al programa vacío, que produce la signatura vacía. Denotamos las estructuras vacías mediante un punto. La segunda regla indica cómo se genera la signatura de un programa compuesto por una lista de definiciones (programa) p seguida de una definición ($\mathsf{fun}\ \rho\ f\ \overline{x : \alpha}\ \overline{s}$). Primero se genera la signatura Σ correspondiente a p, lo cual aparece expresado en la premisa de la regla, y luego se agrega a Σ la asociación del nombre f de la función con la pareja formada por los tipos de los argumentos y el tipo de retorno. La operación de agregado es denotada por una coma. Es importante notar la doble condición lateral para que esta regla pueda ser aplicada. Por un lado, se prohíbe que f ya haya aparecido en el programa y por tanto esté declarada en Σ. Por el otro lado, en este punto se controla que no se usen nombres repetidos entre los argumentos de la función.)

Validez de programa bajo su signatura.

$$\frac{}{\Sigma \vdash . \ \mathsf{ok}} \qquad \frac{\Sigma \vdash p \ \mathsf{ok} \qquad \Sigma; [\overline{x : \alpha}] \vdash_\rho \overline{s} \ \mathsf{ok}}{\Sigma \vdash p \ \ (\mathsf{fun}\ \rho\ f\ \overline{x : \alpha}\ \overline{s}) \quad \mathsf{ok}}$$

(Estas reglas indican cuándo un programa es válido bajo su signatura, procediendo nuevamente por recursión en el programa. Nótese que se va controlando

simplemente cada definición de función en orden de aparición. Al encontrar una definición nueva luego de haber controlado satisfactoriamente las precedentes (segunda regla) se inicializa el contexto de declaraciones de variables de modo que contenga un primer nivel (ámbito) formado por los argumentos de la función y se pasa a controlar el cuerpo de *statements* de la función (segunda premisa).)

Validez de secuencias de instrucciones (*statements*).

$$\frac{}{\Sigma; \Gamma \vdash_\rho . \ \mathsf{ok}}$$

(Esta regla establece que la secuencia vacía de instrucciones es válida en cualquier ambiente.)

$$\frac{\Sigma; \Gamma \vdash e : \alpha \qquad \Sigma; \Gamma \vdash_\rho \overline{s} \ \mathsf{ok}}{\Sigma; \Gamma \vdash_\rho e; \ \overline{s} \ \mathsf{ok}}$$

(Si se tiene una expresión e seguida de *statements* $\overline{s}$ entonces se verifica que la expresión tenga (algún) tipo y luego se continúa verificando la subsiguiente lista $\overline{s}$.)

$$\frac{\Sigma; \Gamma, \overline{x : \alpha} \vdash_\rho \overline{s} \ \mathsf{ok}}{\Sigma; \Gamma \vdash_\rho \alpha \ \overline{x}; \ \overline{s} \ \mathsf{ok}} \ \overline{x} \text{ sin repeticiones y no declaradas en el tope de } \Gamma$$

(Si se tiene una declaración de variables seguida de *statements* $\overline{s}$ entonces se continúa verificando la subsiguiente lista $\overline{s}$, *agregando* las declaraciones de variables al tope del contexto Γ. Ello es representado en la regla por una coma. Debe tenerse presente que el tope del contexto (stack) Γ representa el ámbito corriente. Debe verificarse (condición lateral de la regla) que no haya redeclaración de variables.)

$$\frac{\Sigma; \Gamma \vdash e : \alpha \qquad \Sigma; \Gamma, x : \alpha \vdash_\rho \overline{s} \ \mathsf{ok}}{\Sigma; \Gamma \vdash_\rho \alpha \ x = e; \ \overline{s} \ \mathsf{ok}} \ x \text{ no declarada en el tope de } \Gamma$$

(Si se tiene una declaración de variable con inicialización seguida de *statements* $\overline{s}$ entonces se continúa verificando la subsiguiente lista $\overline{s}$, *agregando* la declaración de la variable en cuestión al tope del contexto Γ. Debe verificarse que la expresión usada en la inicialización sea del tipo declarado para la variable (primera premisa). Asimismo (condición lateral de la regla) la variable no puede estar ya declarada en el ámbito corriente.)

$$\frac{\Sigma; \Gamma \vdash_{\mathsf{void}} \overline{s} \ \mathsf{ok}}{\Sigma; \Gamma \vdash_{\mathsf{void}} \mathsf{return}; \ \overline{s} \ \mathsf{ok}}$$

(Un return vacío (sin expresión) sólo tiene sentido en el contexto de una función cuyo tipo de retorno es void.)

$$\frac{\Sigma; \Gamma \vdash e : \rho \qquad \Sigma; \Gamma \vdash_\rho \overline{s} \text{ ok}}{\Sigma; \Gamma \vdash_\rho \text{return } e; \ \overline{s} \text{ ok}}$$

(En un return de una expresión debe verificarse que ésta tenga el tipo de retorno de la función en la que se encuentra.)

$$\frac{\Sigma; \Gamma \bullet \vdash_\rho \overline{r} \text{ ok} \qquad \Sigma; \Gamma \vdash_\rho \overline{s} \text{ ok}}{\Sigma; \Gamma \vdash_\rho \{\overline{r}\} \ \overline{s} \text{ ok}}$$

(Al aparecer un bloque $\{\overline{r}\}$ debe verificarse la validez de éste abriendo un nuevo ámbito de variables. Para ello se agrega una tabla vacía al tope del stack de declaraciones de variables Γ. Esto es expresado por la notación $\bullet$ en la primera premisa de la regla. Nótese que la segunda premisa expresa que los *statements* $\overline{s}$ subsiguientes al bloque son verificados bajo el contexto de declaraciones de variables que se tenía originalmente (es decir, Γ) descartando el ámbito recién abierto.)

$$\frac{\Sigma; \Gamma \vdash e : \text{bool} \qquad \Sigma; \Gamma \vdash_\rho s \text{ ok} \qquad \Sigma; \Gamma \vdash_\rho \overline{s} \text{ ok}}{\Sigma; \Gamma \vdash_\rho \text{while } (e) \ s \ \ \overline{s} \text{ ok}}$$

(Esta regla es suficientemente explícita.)

$$\frac{\Sigma; \Gamma \vdash e : \text{bool} \qquad \Sigma; \Gamma \vdash_\rho s_1 \text{ ok} \qquad \Sigma; \Gamma \vdash_\rho s_2 \text{ ok} \qquad \Sigma; \Gamma \vdash_\rho \overline{s} \text{ ok}}{\Sigma; \Gamma \vdash_\rho \text{if } (e) \ s_1 \text{ else } s_2 \ \ \overline{s} \text{ ok}}$$

(Ésta también.)

Tipos de expresiones.

$$\frac{}{\Sigma; \Gamma \vdash \text{true} : \text{bool}} \qquad \frac{}{\Sigma; \Gamma \vdash \text{false} : \text{bool}}$$

$$\frac{}{\Sigma; \Gamma \vdash \text{Int } n : \text{int}} \qquad \frac{}{\Sigma; \Gamma \vdash \text{Double } d : \text{double}} \qquad \frac{}{\Sigma; \Gamma \vdash \text{String } w : \text{string}}$$

$$\frac{}{\Sigma; \Gamma \vdash x : \Gamma \ x} \ x \text{ declarada en } \Gamma$$

(En esta regla la notación $\Gamma \ x$ corresponde a la operación de *lookup* del nombre x en el contexto de declaraciones de variables Γ, devolviendo, en caso de que la variable esté disponible, el tipo de ella. Debe recordarse que Γ es un stack de tablas, y que la operación de *lookup* debe, correspondientemente, buscar la declaración de x que aparezca en el ámbito más local posible, es decir en la tabla más cercana al tope de Γ.)

$$\frac{\overline{\Sigma;\Gamma \vdash e : \alpha}}{\Sigma;\Gamma \vdash f(\overline{e}) : \rho} \begin{cases} f \text{ declarada en } \Sigma \\ \Sigma\ f = (\overline{\alpha}, \rho) \\ \overline{e} \text{ del mismo largo que } \overline{\alpha} \end{cases}$$

(Esta regla corresponde a la invocación de una función f. Observemos primero las condiciones laterales. La primera de ellas requiere que f esté presente en la signatura de funciones, es decir, que haya sido definida en el programa en el que se encuentra su invocación. La segunda condición lateral asume que la primera se ha cumplido y entonces especifica cuál es el resultado del *lookup* de f en la tabla Σ. Ése es un par formado por una lista $\overline{\alpha}$ de tipos de los argumentos de la función y un tipo ρ de retorno de la misma. Entonces la lista de expresiones $\overline{e}$ de los argumentos efectivos de la invocación debe tener el mismo largo que la lista $\overline{\alpha}$. Pasando a la premisa de la regla, lo que ella indica es que se debe además aparear las dos listas $\overline{e}$ y $\overline{\alpha}$ y chequear que cada argumento efectivo tenga su correspondiente tipo. El suprarrayado en la premisa de la regla está para indicar que se trata en realidad de una lista de premisas a verificar, una por cada pareja (e, α).)

$$\frac{}{\Sigma;\Gamma \vdash x\texttt{++} : \alpha} \begin{cases} x \text{ declarada en } \Gamma \\ \alpha \in \{\mathsf{int}, \mathsf{double}\} \end{cases} \quad \frac{}{\Sigma;\Gamma \vdash x\texttt{--} : \alpha} \begin{cases} x \text{ declarada en } \Gamma \\ \alpha \in \{\mathsf{int}, \mathsf{double}\} \end{cases}$$

$$\frac{}{\Sigma;\Gamma \vdash \texttt{++}x : \alpha} \begin{cases} x \text{ declarada en } \Gamma \\ \alpha \in \{\mathsf{int}, \mathsf{double}\} \end{cases} \quad \frac{}{\Sigma;\Gamma \vdash \texttt{--}x : \alpha} \begin{cases} x \text{ declarada en } \Gamma \\ \alpha \in \{\mathsf{int}, \mathsf{double}\} \end{cases}$$

(Todas estas reglas y las subsiguientes no necesitan mayor explicación.)

$$\frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{+}e_2 : \alpha}\ \alpha \in \{\mathsf{int}, \mathsf{double}, \mathsf{string}\}$$

$$\frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{-}e_2 : \alpha}\ \alpha \in \{\mathsf{int}, \mathsf{double}\}$$

$$\frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{*}e_2 : \alpha}\ \alpha \in \{\mathsf{int}, \mathsf{double}\}$$

$$\frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{/}e_2 : \alpha}\ \alpha \in \{\mathsf{int}, \mathsf{double}\}$$

$$\frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{<}e_2 : \mathsf{bool}}\ \alpha \in \{\mathsf{int}, \mathsf{double}\} \quad \frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{<=}e_2 : \mathsf{bool}}\ \alpha \in \{\mathsf{int}, \mathsf{double}\}$$

$$\frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{>=}e_2 : \mathsf{bool}}\ \alpha \in \{\mathsf{int}, \mathsf{double}\} \quad \frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{>}e_2 : \mathsf{bool}}\ \alpha \in \{\mathsf{int}, \mathsf{double}\}$$

$$\frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{==}e_2 : \mathsf{bool}}\ \alpha \in \{\mathsf{int}, \mathsf{double}, \mathsf{bool}\}$$

$$\frac{\Sigma;\Gamma \vdash e_1 : \alpha \qquad \Sigma;\Gamma \vdash e_2 : \alpha}{\Sigma;\Gamma \vdash e_1\texttt{!=}e_2 : \mathsf{bool}}\ \alpha \in \{\mathsf{int}, \mathsf{double}, \mathsf{bool}\}$$

$$\frac{\Sigma;\Gamma \vdash e_1 : \mathsf{bool} \qquad \Sigma;\Gamma \vdash e_2 : \mathsf{bool}}{\Sigma;\Gamma \vdash e_1 \texttt{\&\&} e_2 : \mathsf{bool}} \qquad \frac{\Sigma;\Gamma \vdash e_1 : \mathsf{bool} \qquad \Sigma;\Gamma \vdash e_2 : \mathsf{bool}}{\Sigma;\Gamma \vdash e_1 \texttt{||} e_2 : \mathsf{bool}}$$

$$\frac{\Sigma;\Gamma \vdash e : \alpha}{\Sigma;\Gamma \vdash x\texttt{=}e : \Gamma\, x} \begin{cases} x \text{ declarada en } \Gamma \\ \Gamma\, x = \alpha \end{cases}$$