Verificación de Tipos

CI4721 – Lenguajes de Programación II

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

UNIVERSIDAD SIMÓN BOLÍVAR

Sistemas de Tipos

- El **Sistema de Tipos** de un lenguaje define
  - Tipos provistos por el lenguaje – intrínsecos o definibles.
  - Reglas para asignar tipos a las construcciones del lenguaje.
- Reglas Informales – "si ambos operandos son enteros, la operación suma es de tipo entero"
- Reglas Formales – Sistemas Formales para Deducción e Inferencia.
- El **Verificador de Tipos** implanta el Sistema de Tipos.
- Puede haber varios verificadores para un mismo sistema de tipos – decisión del implantador ante ambigüedades de la definición.

El diseño e implantación del componente verificador está íntimamente ligado al sistema de tipos.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cuándo hacer la verificación?

- Verificación estática
  - Durante la compilación.
  - Apoyada en la tabla de símbolos y representación intermedia.
- Verificación dinámica
  - Durante la ejecución.
  - Requiere que el código ejecutable incluya información de tipos.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cuándo hacer la verificación?

- Verificación estática
  - Durante la compilación.
  - Apoyada en la tabla de símbolos y representación intermedia.
- Verificación dinámica
  - Durante la ejecución.
  - Requiere que el código ejecutable incluya información de tipos.
- Un Sistema de Tipos **Sólido** (*Sound Type System*) es aquél susceptible de implantarse en un Verificador completamente estático.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cuándo hacer la verificación?

- Verificación estática
  - Durante la compilación.
  - Apoyada en la tabla de símbolos y representación intermedia.
- Verificación dinámica
  - Durante la ejecución.
  - Requiere que el código ejecutable incluya información de tipos.
- Un Sistema de Tipos **Sólido** (*Sound Type System*) es aquél susceptible de implantarse en un Verificador completamente estático.
- Un lenguaje es de Tipos Fuertes (*Strongly Typed*) si el compilador *garantiza* que la ejecución no tiene errores de tipos.

Aún así, quedarán algunas verificaciones
a tiempo de ejecución.

UNIVERSIDAD SIMÓN BOLÍVAR

Expresiones de Tipos

Meta-lenguaje para los tipos

- **Expresiones de Tipos** – denotan tipos de las construcciones.
- Definición recursiva
  - Casos base – Tipos primitivos.
  - Casos recursivos – Constructores de tipos.
  - Nombres de tipo – si el lenguaje lo permite.
  - Cada lenguaje tendrá sus propias expresiones de tipos.
- Terminan siendo un tipo de datos en el meta-lenguaje – we have to go deeper.

Definiremos *nuestro* sistema de tipos
y sus expresiones de tipos ...

UNIVERSIDAD SIMÓN BOLÍVAR

Expresiones de Tipos

Casos Base – Tipos Primitivos

Todo tipo base será una expresión de tipo.

- Tipos primitivos de nuestro lenguaje
  - **bool**
  - **char**
  - **int**
  - **float**
- **void** – ausencia de valor, aplicable a instrucciones y procedimientos.
- **type_error** – identificar y propagar errores de tipo.

UNIVERSIDAD SIMÓN BOLÍVAR

Expresiones de Tipos

Casos Base – Nombres de Tipos

Todo nombre de tipo será una expresión de tipo.

- Nuestro lenguaje permite definir tipos y asociarles nombres.
- Los nombres denotan la *identidad* del tipo –
  simplifica implantar Equivalencia por Nombres.

UNIVERSIDAD SIMÓN BOLÍVAR

Expresiones de Tipos

Casos Recursivos – Arreglos

Si T es una expresión de tipos,
$array(I, T)$ es una expresión de tipos.

- Arreglo de elementos de tipo T y conjunto de índice I.
- En la expresión

```
var foo : array[1..42] of int
```

el identificador `foo` tendrá tipo

$$array(1..42, \textbf{int})$$

- I podría ser una expresión de tipos o simplemente un conjunto – en nuestro caso manejaremos el conjunto.

Expresiones de Tipos

Casos Recursivos – Producto Cartesiano Anónimo

Si T_1 y T_2 son expresiones de tipos,
$T_1 \times T_2$ es una expresión de tipos.

- Denota el Producto Cartesiano – tuplas
- $\times$ asocia hacia la izquierda.
- En la expresión

```
var foo : ( int , bool , char )
```

el identificador `foo` tendrá tipo

$$(\textbf{int} \times \textbf{bool}) \times \textbf{char}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Expresiones de Tipos

Casos Recursivos – Producto Cartesiano Nombrado

Si $T_1, \ldots, T_k$ son expresiones de tipos y
$N_1, \ldots N_k$ son nombres para cada uno,
$record((N_1 \times T_1) \times \cdots \times (N_k \times T_k))$
es una expresión de tipos.

- Registro – producto cartesiano con nombres para cada componente.
- El primer componente de cada tupla es el nombre del campo.
- En la expresión

```
type grok = struct {
              foo char,
              bar array[1..42] of bool
            };
```

el identificador `grok` tendrá tipo

$$record((\textbf{foo} \times \textbf{char}) \times (\textbf{bar} \times array(1..42, \textbf{bool})))$$

UNIVERSIDAD SIMÓN BOLÍVAR

Expresiones de Tipos

Casos Recursivos – Apuntadores

Si T es una expresión de tipos,
pointer(T) es una expresión de tipos.

- "Apuntador a un valor de tipo T"
- En la expresión

```
var foo = *grok;
```

el identificador `foo` tendrá tipo

pointer(**grok**)

Expresiones de Tipos

Casos Recursivos – Funciones

Si D y R son expresiones de tipos,
$D \to R$ es una expresión de tipos.

- En la expresión

```
fun qux( foo, bar : int, baz : real ) : *char
```

el identificador `qux` tendrá tipo

$$((\textbf{int} \times \textbf{int}) \times \textbf{real}) \to \textit{pointer}(\textbf{char})$$

- Noten que *todas* las funciones son de **un** argumento.

UNIVERSIDAD SIMÓN BOLÍVAR

Expresiones de Tipos

Casos Recursivos – Funciones

Si D y R son expresiones de tipos,
$D \to R$ es una expresión de tipos.

- En la expresión

```
fun qux( foo, bar : int, baz : real ) : *char
```

el identificador `qux` tendrá tipo

$$((\textbf{int} \times \textbf{int}) \times \textbf{real}) \to \mathit{pointer}(\textbf{char})$$

- Noten que *todas* las funciones son de **un** argumento.
- Hay quien la escribe

$$\mathit{fun}(((\textbf{int} \times \textbf{int}) \times \textbf{real}), \mathit{pointer}(\textbf{char}))$$

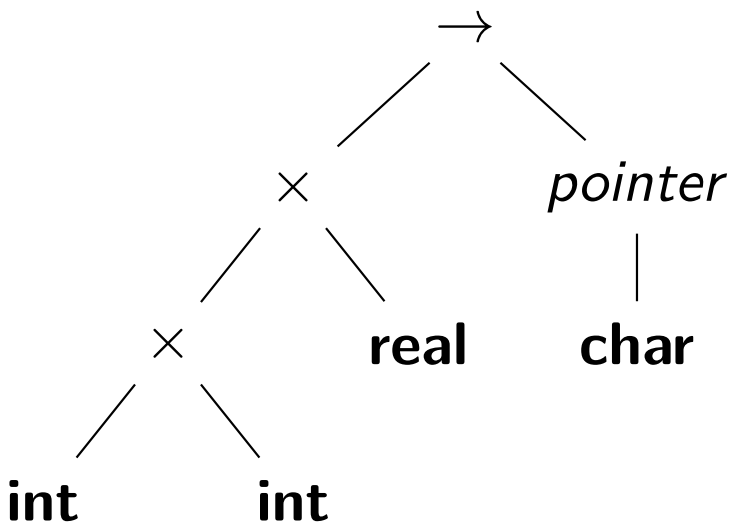
y me entristece su vida carente de la emoción de UTF-8

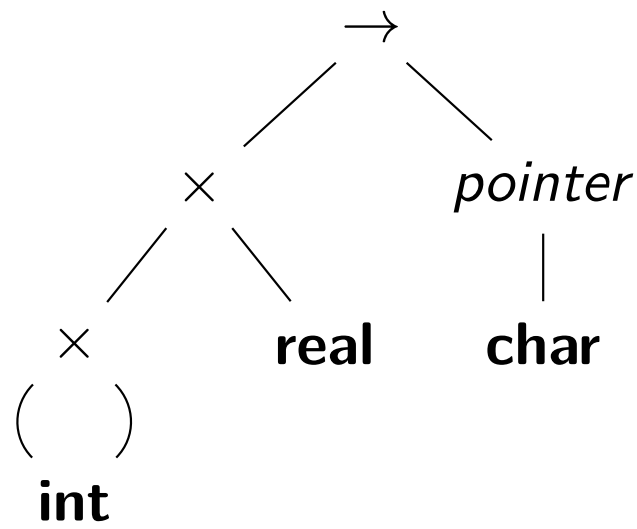
UNIVERSIDAD SIMÓN BOLÍVAR

Representación de Expresiones de Tipos

Son grafos

$$((\textbf{int} \times \textbf{int}) \times \textbf{real}) \rightarrow \textit{pointer}(\textbf{char})$$





Aprovechar los Esquemas de Traducción para construirlos.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos Dirigido por Sintaxis

La gramática – Programas y Declaraciones

$P \rightarrow D \ ; \ E$
$D \rightarrow D \ ; \ D$
$D \rightarrow \textbf{id} : T$
$T \rightarrow \textbf{char}$
$T \rightarrow \textbf{int}$
$T \rightarrow \textbf{bool}$
$T \rightarrow \textbf{real}$
$T \rightarrow * \ T_1$
$T \rightarrow \textbf{array} \ [\ \textbf{num} \ ] \ \textbf{of} \ T_1$

- Programa – lista de declaraciones seguido de expresiones.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos Dirigido por Sintaxis

Declaración – Expresiones de tipos primitivos

$P \rightarrow D \ ; \ E$	
$D \rightarrow D \ ; \ D$	
$D \rightarrow \textbf{id} : T$	
$T \rightarrow \textbf{char}$	$\{ \ T.type \leftarrow \textbf{char} \ \}$
$T \rightarrow \textbf{int}$	$\{ \ T.type \leftarrow \textbf{int} \ \}$
$T \rightarrow \textbf{bool}$	$\{ \ T.type \leftarrow \textbf{bool} \ \}$
$T \rightarrow \textbf{real}$	$\{ \ T.type \leftarrow \textbf{real} \ \}$
$T \rightarrow * \ T_1$	
$T \rightarrow \textbf{array} \ [\ \textbf{num} \ ] \ \textbf{of} \ T_1$	

- Programa – lista de declaraciones seguido de expresiones.
- *type* es sintetizado – grafo para la expresión de tipo.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos Dirigido por Sintaxis

Declaración – Expresiones de tipo apuntador

$P \rightarrow D \ ; \ E$	
$D \rightarrow D \ ; \ D$	
$D \rightarrow \textbf{id} : T$	
$T \rightarrow \textbf{char}$	$\{ \ T.type \leftarrow \textbf{char} \ \}$
$T \rightarrow \textbf{int}$	$\{ \ T.type \leftarrow \textbf{int} \ \}$
$T \rightarrow \textbf{bool}$	$\{ \ T.type \leftarrow \textbf{bool} \ \}$
$T \rightarrow \textbf{real}$	$\{ \ T.type \leftarrow \textbf{real} \ \}$
$T \rightarrow * \ T_1$	$\{ \ T.type \leftarrow pointer(T_1.type) \ \}$
$T \rightarrow \textbf{array} \ [\ \textbf{num} \ ] \ \textbf{of} \ T_1$	

- Programa – lista de declaraciones seguido de expresiones.
- *type* es sintetizado – grafo para la expresión de tipo.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos Dirigido por Sintaxis

Declaración – Expresiones de tipo arreglo

$P \rightarrow D \ ; \ E$	
$D \rightarrow D \ ; \ D$	
$D \rightarrow \textbf{id} : T$	
$T \rightarrow \textbf{char}$	$\{ \ T.type \leftarrow \textbf{char} \ \}$
$T \rightarrow \textbf{int}$	$\{ \ T.type \leftarrow \textbf{int} \ \}$
$T \rightarrow \textbf{bool}$	$\{ \ T.type \leftarrow \textbf{bool} \ \}$
$T \rightarrow \textbf{real}$	$\{ \ T.type \leftarrow \textbf{real} \ \}$
$T \rightarrow * \ T_1$	$\{ \ T.type \leftarrow pointer(T_1.type) \ \}$
$T \rightarrow \textbf{array} \ [\ \textbf{num} \ ] \ \textbf{of} \ T_1$	$\{ \ T.type \leftarrow array(0..\textbf{num}.val - 1, T_1.type) \ \}$

- Programa – lista de declaraciones seguido de expresiones.
- *type* es sintetizado – grafo para la expresión de tipo.
- Los arreglos tienen base cero.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos Dirigido por Sintaxis

Declaración – Asociar la expresión al identificador

$P \rightarrow D \ ; \ E$	
$D \rightarrow D \ ; \ D$	
$D \rightarrow \textbf{id} : T$	$\{ \ addtype(\textbf{id}.lexema, T.type) \ \}$
$T \rightarrow \textbf{char}$	$\{ \ T.type \leftarrow \textbf{char} \ \}$
$T \rightarrow \textbf{int}$	$\{ \ T.type \leftarrow \textbf{int} \ \}$
$T \rightarrow \textbf{bool}$	$\{ \ T.type \leftarrow \textbf{bool} \ \}$
$T \rightarrow \textbf{real}$	$\{ \ T.type \leftarrow \textbf{real} \ \}$
$T \rightarrow * \ T_1$	$\{ \ T.type \leftarrow pointer(T_1.type) \ \}$
$T \rightarrow \textbf{array} \ [\ \textbf{num} \ ] \ \textbf{of} \ T_1$	$\{ \ T.type \leftarrow array(0..\textbf{num}.val - 1, T_1.type) \ \}$

- Programa – lista de declaraciones seguido de expresiones.
- *type* es sintetizado – grafo para la expresión de tipo.
- Los arreglos tienen base cero.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones Simples

La gramática

$E \rightarrow$ **literal**
$E \rightarrow$ **num**
$E \rightarrow$ **num.num**
$E \rightarrow$ **true**
$E \rightarrow$ **false**
$E \rightarrow$ **id**

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones Simples

Traducción – Tipos de las expresiones básicas

$E \rightarrow$ **literal**	$\{\ E.type \leftarrow$ **char** $\}$
$E \rightarrow$ **num**	$\{\ E.type \leftarrow$ **int** $\}$
$E \rightarrow$ **num.num**	$\{\ E.type \leftarrow$ **real** $\}$
$E \rightarrow$ **true**	$\{\ E.type \leftarrow$ **bool** $\}$
$E \rightarrow$ **false**	$\{\ E.type \leftarrow$ **bool** $\}$
$E \rightarrow$ **id**	

- **literal** es el *token* para caracteres – '*' ASCII 42.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones Simples

Traducción – Tipo dado por un identificador

$E \rightarrow$ **literal**	$\{\ E.type \leftarrow$ **char** $\}$
$E \rightarrow$ **num**	$\{\ E.type \leftarrow$ **int** $\}$
$E \rightarrow$ **num.num**	$\{\ E.type \leftarrow$ **real** $\}$
$E \rightarrow$ **true**	$\{\ E.type \leftarrow$ **bool** $\}$
$E \rightarrow$ **false**	$\{\ E.type \leftarrow$ **bool** $\}$
$E \rightarrow$ **id**	$\{\ E.type \leftarrow lookup($**id**$.lexema)\ \}$

- **literal** es el *token* para caracteres – '*' ASCII 42.
- Si **id** no tiene tipo asociado, *lookup* retorna **type_error**.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones complejas

La gramática

$E \rightarrow E_1 + E_2$

$E \rightarrow E_1$ **and** E_2

$E \rightarrow E_1 < E_2$

$E \rightarrow E_1$ **mod** E_2

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones complejas

Traducción – Operador binario aritmético

$E \rightarrow E_1 + E_2 \quad \left\{ E.type \leftarrow \quad \begin{array}{l} \textbf{if } E_1.type = E_2.type \\ \textbf{then } E_1.type \textbf{ else type\_error} \end{array} \right\}$

$E \rightarrow E_1$ **and** E_2

$E \rightarrow E_1 < E_2$

$E \rightarrow E_1$ **mod** E_2

- No contemplamos conversión ni coerción por ahora...

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones complejas

Traducción – Operador binario booleano

$E \rightarrow E_1 + E_2$ $\quad \left\{ E.type \leftarrow \text{ if } E_1.type = E_2.type \text{ then } E_1.type \text{ else type\_error} \right\}$

$E \rightarrow E_1 \textbf{ and } E_2$ $\quad \left\{ E.type \leftarrow \text{ if } (E_1.type = \textbf{bool} \wedge E_2.type = \textbf{bool}) \text{ then bool else type\_error} \right\}$

$E \rightarrow E_1 < E_2$

$E \rightarrow E_1 \textbf{ mod } E_2$

- No contemplamos conversión ni coerción por ahora...

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones complejas

Traducción – Operador binario booleano

$$
\begin{array}{ll}
E \rightarrow E_1 + E_2 & \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } E_1.type = E_2.type \\ \textbf{then } E_1.type \textbf{ else type\_error} \end{array} \right\} \\
E \rightarrow E_1 \textbf{ and } E_2 & \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_1.type = \textbf{bool} \wedge E_2.type = \textbf{bool}) \\ \textbf{then bool else type\_error} \end{array} \right\} \\
E \rightarrow E_1 < E_2 & \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_1.type = E_2.type \wedge E_1.type \neq \textbf{type\_error}) \\ \textbf{then bool else type\_error} \end{array} \right\} \\
E \rightarrow E_1 \textbf{ mod } E_2 &
\end{array}
$$

- No contemplamos conversión ni coerción por ahora...
- Estoy suponiendo que todos los tipos son "comparables"

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones complejas

Traducción – Operador binario restrictivo

$$
\begin{array}{ll}
E \rightarrow E_1 + E_2 & \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } E_1.type = E_2.type \\ \textbf{then } E_1.type \textbf{ else type\_error} \end{array} \right\} \\
E \rightarrow E_1 \textbf{ and } E_2 & \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_1.type = \textbf{bool} \wedge E_2.type = \textbf{bool}) \\ \textbf{then bool else type\_error} \end{array} \right\} \\
E \rightarrow E_1 < E_2 & \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_1.type = E_2.type \wedge E_1.type \neq \textbf{type\_error}) \\ \textbf{then bool else type\_error} \end{array} \right\} \\
E \rightarrow E_1 \textbf{ mod } E_2 & \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_1.type = E_2.type \wedge E_1.type = \textbf{int}) \\ \textbf{then int else type\_error} \end{array} \right\}
\end{array}
$$

- No contemplamos conversión ni coerción por ahora...
- Estoy suponiendo que todos los tipos son "comparables"

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones complejas

La gramática

$$E \rightarrow E_1 \; [\; E_2 \;]$$

$$E \rightarrow * \; E_1$$

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones complejas

Traducción – Acceso a un arreglo

$$E \rightarrow E_1\ [\ E_2\ ] \quad \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_2.type = \textbf{int} \wedge E_1.type = array(s,t)) \\ \textbf{then } t \textbf{ else type\_error} \end{array} \right\}$$

$$E \rightarrow * E_1$$

- Verificar que el tipo de E_2 siempre estará en el rango del tipo para el índice no *siempre* es posible a tiempo de compilación.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Expresiones complejas

Traducción – Acceso a un apuntador

$$E \rightarrow E_1\ [\ E_2\ ] \quad \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_2.type = \textbf{int} \wedge E_1.type = array(s,t)) \\ \textbf{then } t \textbf{ else type\_error} \end{array} \right\}$$

$$E \rightarrow * E_1 \quad \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } E_1.type = pointer(t) \\ \textbf{then } t \textbf{ else type\_error} \end{array} \right\}$$

- Verificar que el tipo de E_2 siempre estará en el rango del tipo para el índice no *siempre* es posible a tiempo de compilación.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Instrucciones

La gramática

$S \rightarrow$ **id** $:= E$

$S \rightarrow$ **if** E **then** S_1

$S \rightarrow$ **while** E **do** S_1

$S \rightarrow S_1 \ ; \ S_2$

- Supondremos que las instrucciones **no** tienen valor.
- Cambiamos la primera producción por $P \rightarrow D \ ; \ S$

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Instrucciones

Traducción – Asignación

$S \rightarrow \textbf{id} := E$ $\quad \left\{ S.type \leftarrow \begin{array}{l} \textbf{if id}.type = E.type \\ \textbf{then void else type\_error} \end{array} \right\}$

$S \rightarrow \textbf{if } E \textbf{ then } S_1$

$S \rightarrow \textbf{while } E \textbf{ do } S_1$

$S \rightarrow S_1 \ ; \ S_2$

- Supondremos que las instrucciones **no** tienen valor.
- Cambiamos la primera producción por $P \rightarrow D \ ; \ S$

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Instrucciones

Traducción – Selector

$S \rightarrow \textbf{id} := E$

$$\left\{ S.type \leftarrow \begin{array}{l} \textbf{if } \textbf{id}.type = E.type \\ \textbf{then void else type\_error} \end{array} \right\}$$

$S \rightarrow \textbf{if } E \textbf{ then } S_1$

$$\left\{ S.type \leftarrow \begin{array}{l} \textbf{if } E.type = \textbf{bool} \\ \textbf{then } S_1.type \textbf{ else type\_error} \end{array} \right\}$$

$S \rightarrow \textbf{while } E \textbf{ do } S_1$

$S \rightarrow S_1 \ ; \ S_2$

- Supondremos que las instrucciones **no** tienen valor.
- Cambiamos la primera producción por $P \rightarrow D \ ; \ S$
- Se propagan los tipos hacia la instrucción que envuelve.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Instrucciones

Traducción – Iteración

$S \rightarrow$ **id** $:= E$

$$\left\{ S.type \leftarrow \begin{array}{l} \textbf{if id}.type = E.type \\ \textbf{then void else type\_error} \end{array} \right\}$$

$S \rightarrow$ **if** E **then** S_1

$$\left\{ S.type \leftarrow \begin{array}{l} \textbf{if } E.type = \textbf{bool} \\ \textbf{then } S_1.type \textbf{ else type\_error} \end{array} \right\}$$

$S \rightarrow$ **while** E **do** S_1

$$\left\{ S.type \leftarrow \begin{array}{l} \textbf{if } E.type = \textbf{bool} \\ \textbf{then } S_1.type \textbf{ else type\_error} \end{array} \right\}$$

$S \rightarrow S_1 \ ; \ S_2$

- Supondremos que las instrucciones **no** tienen valor.
- Cambiamos la primera producción por $P \rightarrow D \ ; \ S$
- Se propagan los tipos hacia la instrucción que envuelve.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Instrucciones

Traducción – Secuenciación

$$S \rightarrow \textbf{id} := E \quad \left\{ S.type \leftarrow \textbf{if } \textbf{id}.type = E.type \textbf{ then void else type\_error} \right\}$$

$$S \rightarrow \textbf{if } E \textbf{ then } S_1 \quad \left\{ S.type \leftarrow \textbf{if } E.type = \textbf{bool} \textbf{ then } S_1.type \textbf{ else type\_error} \right\}$$

$$S \rightarrow \textbf{while } E \textbf{ do } S_1 \quad \left\{ S.type \leftarrow \textbf{if } E.type = \textbf{bool} \textbf{ then } S_1.type \textbf{ else type\_error} \right\}$$

$$S \rightarrow S_1 \ ; \ S_2 \quad \left\{ S.type \leftarrow \textbf{if } (S_1.type = \textbf{void} \wedge S_2.type = \textbf{void}) \textbf{ then void else type\_error} \right\}$$

- Supondremos que las instrucciones **no** tienen valor.
- Cambiamos la primera producción por $P \rightarrow D \ ; \ S$
- Se propagan los tipos hacia la instrucción que envuelve.

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Funciones

La gramática

$$T \rightarrow T_1 -> T_2$$

$$E \rightarrow E_1 \ (\ E_2 \ )$$

Verificador de Tipos – Funciones

Traducción – Declaración

$$T \rightarrow T_1 \; - > \; T_2 \qquad \{T.type := T_1.type \rightarrow T_2.type\}$$

$$E \rightarrow E_1 \; (\; E_2 \;)$$

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Funciones

Traducción – Aplicación

$$T \rightarrow T_1 \; -> \; T_2 \qquad \{T.type := T_1.type \rightarrow T_2.type\}$$

$$E \rightarrow E_1 \; (\; E_2 \;) \qquad \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_1.type = s \rightarrow t \wedge E_2.type = s) \\ \textbf{then } t \textbf{ else type\_error} \end{array} \right\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Verificador de Tipos – Funciones

Traducción – Extensión

$$T \rightarrow T_1 -> T_2 \quad \{T.type := T_1.type \rightarrow T_2.type\}$$

$$E \rightarrow E_1 \ (\ E_2 \ ) \quad \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_1.type = s \rightarrow t \wedge E_2.type = s) \\ \textbf{then } t \textbf{ else type\_error} \end{array} \right\}$$

- La extensión para múltiples argumentos *sin* curryficación
  - T_1 utiliza un tipo tupla para establecer el tipo de los formales.
  - Cambiar la regla de aplicación por $E \rightarrow E_1(L)$ donde L determina el tipo tupla de la lista de los actuales.
- ¿Qué hay que hacer para que funcione con curryficación?

UNIVERSIDAD SIMÓN BOLÍVAR

Sistema Formal de Tipos

Componentes

- Metalenguaje para razonar sobre los tipos de un lenguaje L particular.

Sistema Formal de Tipos

Componentes

- Metalenguaje para razonar sobre los tipos de un lenguaje L particular.
- El **Ambiente** ρ (*Environment*)
  - Conjunto de pares $< nombre, tipo >$ – componentes pertenecen a L.
  - Se puede consultar para determinar el tipo asociado a un nombre

$$\rho(\upsilon) = \tau$$

UNIVERSIDAD SIMÓN BOLÍVAR

Sistema Formal de Tipos

Componentes

- Metalenguaje para razonar sobre los tipos de un lenguaje L particular.
- El **Ambiente** ρ (*Environment*)
  - Conjunto de pares $< nombre, tipo >$ – componentes pertenecen a L.
  - Se puede consultar para determinar el tipo asociado a un nombre

$$\rho(\upsilon) = \tau$$

- Un **Juicio** (*Type Judgment*)

$$e : \tau$$

establece que la expresión $e \in L$ tiene el tipo τ

UNIVERSIDAD SIMÓN BOLÍVAR

Sistema Formal de Tipos

Componentes

- Metalenguaje para razonar sobre los tipos de un lenguaje L particular.
- El **Ambiente** ρ (*Environment*)
  - Conjunto de pares $< nombre, tipo >$ – componentes pertenecen a L.
  - Se puede consultar para determinar el tipo asociado a un nombre

$$\rho(\upsilon) = \tau$$

- Un **Juicio** (*Type Judgment*)

$$e : \tau$$

establece que la expresión $e \in L$ tiene el tipo τ

- El Sistema de Tipos dispone de **Reglas** (*Type Rules*) para asignar un tipo a cada construcción sintáctica de L.

UNIVERSIDAD SIMÓN BOLÍVAR

Sistema Formal de Tipos

De premisas a conclusión

- Un juicio puede ser establecido por el ambiente

$$\frac{\rho(v) = \tau}{\rho \vdash v : \tau}$$

Sistema Formal de Tipos

De premisas a conclusión

- Un juicio puede ser establecido por el ambiente

$$\frac{\rho(v) = \tau}{\rho \vdash v : \tau}$$

- Establecido por las reglas de tipos de *L*

$$\frac{\rho(v) = \tau \qquad \rho \vdash e : \tau}{\rho \vdash v := e : \textbf{void}}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Sistema Formal de Tipos

De premisas a conclusión

- Un juicio puede ser establecido por el ambiente

$$\frac{\rho(v) = \tau}{\rho \vdash v : \tau}$$

- Establecido por las reglas de tipos de *L*

$$\frac{\rho(v) = \tau \qquad \rho \vdash e : \tau}{\rho \vdash v := e : \textbf{void}}$$

Verificar los tipos es equivalente a demostrar teoremas.

UNIVERSIDAD SIMÓN BOLÍVAR

Sistema Formal de Tipos

Verificación de tipos

Sea el ambiente

$$\rho = \{< \textbf{foo}, \textbf{int} >, < \textbf{bar}, \textbf{int} >, < 42, \textbf{int} >\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Sistema Formal de Tipos

Verificación de tipos

Sea el ambiente

$$\rho = \{< \textbf{foo}, \textbf{int} >, < \textbf{bar}, \textbf{int} >, < 42, \textbf{int} >\}$$

Usando los juicios antes definidos y agregando el juicio adicional

$$\frac{\rho \vdash e_1 : \textbf{int} \qquad \rho \vdash e_2 : \textbf{int}}{\rho \vdash \ e_1 + e_2 : \textbf{int}}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Sistema Formal de Tipos

Verificación de tipos

Sea el ambiente

$$\rho = \{< \textbf{foo}, \textbf{int} >, < \textbf{bar}, \textbf{int} >, < 42, \textbf{int} >\}$$

Usando los juicios antes definidos y agregando el juicio adicional

$$\frac{\rho \vdash e_1 : \textbf{int} \qquad \rho \vdash e_2 : \textbf{int}}{\rho \vdash \ e_1 + e_2 : \textbf{int}}$$

podemos verificar el tipo de la expresión **foo := bar + 42** con el juicio

$$\frac{\rho(\textbf{foo}) = \textbf{int} \qquad \dfrac{\dfrac{\rho(\textbf{bar}) = \textbf{int}}{\rho \vdash \textbf{bar} : \textbf{int}} \qquad \dfrac{\rho(\textbf{42}) = \textbf{int}}{\rho \vdash \textbf{42} : \textbf{int}}}{\rho \vdash \textbf{bar} + \textbf{42} : \textbf{int}}}{\rho \vdash \ \textbf{foo} := \textbf{bar} + \textbf{42} : \textbf{void}}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Formalización de la Traducción

Verificación de Instrucciones – Asignación

El juicio

$$\frac{\rho(v) = \tau \qquad \rho \vdash e : \tau}{\rho \vdash v := e : \textbf{void}}$$

corresponde con el esquema de traducción

$$S \rightarrow \textbf{id} := E \quad \left\{ \begin{array}{ll} S.type \leftarrow & \textbf{if id}.type = E.type \\ & \textbf{then void else type\_error} \end{array} \right\}$$

Formalización de la Traducción

Verificación de Instrucciones – Selector

El juicio

$$\frac{\rho(e) = \textbf{bool} \qquad \rho \vdash s : \tau}{\rho \vdash \textbf{ if } e \textbf{ then } s : \tau}$$

corresponde con el esquema de traducción

$$S \rightarrow \textbf{ if } E \textbf{ then } S_1 \quad \left\{ S.type \leftarrow \begin{array}{l} \textbf{if } E.type = \textbf{bool} \\ \textbf{then } S_1.type \textbf{ else type\_error} \end{array} \right\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Formalización de la Traducción

Verificación de Instrucciones – Iteración

El juicio

$$\frac{\rho(e) = \textbf{bool} \qquad \rho \vdash s : \tau}{\rho \vdash \textbf{while } e \textbf{ do } s : \tau}$$

corresponde con el esquema de traducción

$$S \rightarrow \textbf{while } E \textbf{ do } S_1 \quad \left\{ \begin{array}{ll} S.type \leftarrow & \textbf{if } E.type = \textbf{bool} \\ & \textbf{then } S_1.type \textbf{ else type\_error} \end{array} \right\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Formalización de la Traducción

Verificación de Instrucciones – Secuenciación

El juicio

$$\frac{\rho \vdash s_1 : \textbf{void} \qquad \rho \vdash s_2 : \textbf{void}}{\rho \vdash s_1 \ ; \ s_2 : \textbf{void}}$$

corresponde con el esquema de traducción

$$S \rightarrow S_1 \ ; \ S_2 \quad \left\{ S.type \leftarrow \begin{array}{l} \textbf{if } (S_1.type = \textbf{void} \wedge S_2.type = \textbf{void}) \\ \textbf{then void else type\_error} \end{array} \right\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Formalización de la Traducción

Verificación de Expresiones – Expresiones simples

El juicio

$$\frac{\rho(v) \;=\; \tau}{\rho \vdash \; v : \tau}$$

corresponde con todos los esquemas de traducción

$E \rightarrow$ **literal**	$\{\ E.type \leftarrow$ **char** $\}$
$E \rightarrow$ **num**	$\{\ E.type \leftarrow$ **int** $\}$
$E \rightarrow$ **num.num**	$\{\ E.type \leftarrow$ **real** $\}$
$E \rightarrow$ **true**	$\{\ E.type \leftarrow$ **bool** $\}$
$E \rightarrow$ **false**	$\{\ E.type \leftarrow$ **bool** $\}$
$E \rightarrow$ **id**	$\{\ E.type \leftarrow lookup($**id**$.lexema)\ \}$

sustituyendo v por el lado derecho correspondiente y τ por el tipo.

Formalización de la Traducción

Verificación de Expresiones – Expresiones complejas

El juicio

$$\frac{\rho \vdash e_1 : \textbf{bool} \qquad \rho \vdash e_2 : \textbf{bool}}{\rho \vdash \ e_1 \ \textbf{and} \ e_2 : \textbf{bool}}$$

corresponde con el esquema de traducción

$$E \rightarrow E_1 \ \textbf{and} \ E_2 \quad \left\{ \begin{array}{ll} E.type \leftarrow & \textbf{if } (E_1.type = \textbf{bool} \wedge E_2.type = \textbf{bool}) \\ & \textbf{then bool else type\_error} \end{array} \right\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Formalización de la Traducción

Verificación de Expresiones – Expresiones complejas

El juicio

$$\frac{\rho \vdash e_1 : \tau \qquad \rho \vdash e_2 : \tau}{\rho \vdash e_1 < e_2 : \textbf{bool}}$$

corresponde con el esquema de traducción

$$E \to E_1 < E_2 \quad \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } (E_1.type = E_2.type \wedge E_1.type \neq \textbf{type\_error}) \\ \textbf{then bool else type\_error} \end{array} \right\}$$

con la misma suposición de que todos los tipos son "comparables".

UNIVERSIDAD SIMÓN BOLÍVAR

Formalización de la Traducción

Verificación de Expresiones – Acceso a un arreglo

El juicio

$$\frac{\rho \vdash e_1 : \mathit{array}(s, \tau) \qquad \rho \vdash e_2 : \textbf{int}}{\rho \vdash \ e_1 \ [\ e_2\ ] : \tau}$$

corresponde con el esquema de traducción

$$E \rightarrow E_1\ [\ E_2\ ] \quad \left\{ E.\mathit{type} \leftarrow \begin{array}{l} \textbf{if } (E_2.\mathit{type} = \textbf{int} \wedge E_1.\mathit{type} = \mathit{array}(s,t)) \\ \textbf{then } t \textbf{ else type\_error} \end{array} \right\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Formalización de la Traducción

Verificación de Expresiones – Acceso a un apuntador

El juicio

$$\frac{\rho \vdash e : pointer(\tau)}{\rho \vdash\ * e : \tau}$$

corresponde con el esquema de traducción

$$E \rightarrow * E_1 \quad \left\{ E.type \leftarrow \begin{array}{l} \textbf{if } E_1.type = pointer(t) \\ \textbf{then } t \textbf{ else type\_error} \end{array} \right\}$$

Formalización de la Traducción

Verificación de Expresiones – Aplicación de funciones

El juicio

$$\frac{\rho \vdash e_1 : \mathit{fun}(\sigma, \tau) \qquad \rho \vdash e_2 : \sigma}{\rho \vdash \ e_1 \ (\ e_2 \ ) : \tau}$$

corresponde con el esquema de traducción

$$E \rightarrow E_1 \ (\ E_2 \ ) \quad \left\{ E.\mathit{type} \leftarrow \quad \begin{array}{l} \textbf{if } (E_1.\mathit{type} = s \rightarrow t \wedge E_2.\mathit{type} = s) \\ \textbf{then } t \textbf{ else type\_error} \end{array} \right\}$$

usando *fun* en lugar de $\rightarrow$, por supuesto.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo preparar el ambiente?

- El ambiente inicialmente está vacío.
- El sistema de verificación puede *agregar* información al ambiente empleando una regla de la forma

$$\frac{\textbf{true}}{\rho, < \textbf{id}, \tau > \ \vdash \ \textbf{id} : \tau}$$

que corresponde al esquema

$$D \rightarrow \textbf{id} : T \quad \{ \ addtype(\textbf{id}.lexema, T.type) \ \}$$

- En la conclusión de la regla
  - $\rho, < \textbf{id}, \tau >$ denota el *nuevo* ambiente después de agregar la tupla.
  - El primer **:** es del lenguaje y el segundo del meta-lenguaje.

UNIVERSIDAD SIMÓN BOLÍVAR

Bibliografía

- [*Aho*]
  - Capítulo 6
  - Ejercicios 6.1 a 6.10
- Agregue producciones y reglas de traducción para
  - Completar los operadores aritméticos y booleanos, pero siendo más preciso en las condiciones sobre los tipos.
  - Permitir el uso de tuplas.
  - Construir DAGs en lugar de árboles.
- Escriba las reglas de tipos para las producciones adicionales.
- [*Cardelli*]
  Type Systems
- WikiPedia – Type System
- WikiPedia – Type Rules

UNIVERSIDAD SIMÓN BOLÍVAR