Verificación de Tipos

CI4721 – Lenguajes de Programación II

Ernesto Hernández-Novich
<emhn@usb.ve>

Universidad "Simón Bolívar"

UNIVERSIDAD SIMÓN BOLÍVAR

Funciones Polimórficas

- Una **Función Polimórfica** puede ser ejecutada con argumentos de tipos diferentes.
  - No se trata de sobrecarga – es **una** función.
  - No se trata de *templates* (C++, Java) – no hay copia de código.
  - Polimorfismo paramétrico (Haskell, ML).

UNIVERSIDAD SIMÓN BOLÍVAR

Funciones Polimórficas

- Una **Función Polimórfica** puede ser ejecutada con argumentos de tipos diferentes.
  - No se trata de sobrecarga – es **una** función.
  - No se trata de *templates* (C++, Java) – no hay copia de código.
  - Polimorfismo paramétrico (Haskell, ML).
- Casi cualquier lenguaje tiene alguna función u operador polimórfico.
  - Acceder a posiciones de arreglos – sirve con arreglos de cualquier tipo.
  - Obtener la dirección de un objeto – la obtiene para cualquier objeto.
  - Seguir apuntadores – opera con apuntadores a cualquier cosa.

UNIVERSIDAD SIMÓN BOLÍVAR

Funciones Polimórficas

- Una **Función Polimórfica** puede ser ejecutada con argumentos de tipos diferentes.
  - No se trata de sobrecarga – es **una** función.
  - No se trata de *templates* (C++, Java) – no hay copia de código.
  - Polimorfismo paramétrico (Haskell, ML).
- Casi cualquier lenguaje tiene alguna función u operador polimórfico.
  - Acceder a posiciones de arreglos – sirve con arreglos de cualquier tipo.
  - Obtener la dirección de un objeto – la obtiene para cualquier objeto.
  - Seguir apuntadores – opera con apuntadores a cualquier cosa.

¿Cómo verificar tipos de manera general
si el *usuario* puede definir funciones polimórficas?

UNIVERSIDAD SIMÓN BOLÍVAR

¿Por qué son interesantes?

Longitud de una lista – sin polimorfismo

```
type link = *cell;
     cell = record
       info : integer;
       next : link
     end;
function length (l : link) : integer;
var len : integer;
begin
  len := 0;
  while l <> nil do begin
    len := len + 1;
    l := *l.next
  end;
  return len
end;
```

¿Por qué son interesantes?

Longitud de una lista – sin polimorfismo

```
type link = *cell;
     cell = record
       info : integer;
       next : link
     end;
function length (l : link) : integer;
var len : integer;
begin
  len := 0;
  while l <> nil do begin
    len := len + 1;
    l := *l.next
  end;
  return len
end;
```

Hace falta una función `length` por *cada* tipo de lista.

¿Por qué son interesantes?

Longitud de una lista – con polimorfismo

```
function length(x) =
  if null(x) then 0
             else length(tail(x)) + 1
```

`length` no necesita conocer el tipo contenido por la lista – ¡no lo usa!

UNIVERSIDAD SIMÓN BOLÍVAR

¿Por qué son interesantes?

What kind of sorcery is this?

```
> length([42,17,69])
3
> length(["foo","bar","baz","qux"])
4
> length([])
0
```

- La *única* función `length` se comporta como si tuviera los tipos

¿Por qué son interesantes?

What kind of sorcery is this?

```
> length([42,17,69])
3
> length(["foo","bar","baz","qux"])
4
> length([])
0
```

- La *única* función `length` se comporta como si tuviera los tipos
  - $list(\textbf{int}) \rightarrow \textbf{int}$

UNIVERSIDAD SIMÓN BOLÍVAR

¿Por qué son interesantes?

What kind of sorcery is this?

```
> length([42,17,69])
3
> length(["foo","bar","baz","qux"])
4
> length([])
0
```

- La *única* función `length` se comporta como si tuviera los tipos
  - $list(\textbf{int}) \rightarrow \textbf{int}$
  - $list(\textbf{string}) \rightarrow \textbf{int}$

UNIVERSIDAD SIMÓN BOLÍVAR

¿Por qué son interesantes?

What kind of sorcery is this?

```
> length([42,17,69])
3
> length(["foo","bar","baz","qux"])
4
> length([])
0
```

- La *única* función `length` se comporta como si tuviera los tipos
  - $list(\textbf{int}) \rightarrow \textbf{int}$
  - $list(\textbf{string}) \rightarrow \textbf{int}$
  - $list(whatever) \rightarrow \textbf{int}$

What do you mean “whatever”?

UNIVERSIDAD SIMÓN BOLÍVAR

Variables de Tipo

Whatever floats your boat...

- En la última aplicación de `length` no solamente no está claro sino que *no importa* cuál es el contenido de la lista – cualquiera funcionaría.

Variables de Tipo

Whatever floats your boat...

- En la última aplicación de `length` no solamente no está claro sino que *no importa* cuál es el contenido de la lista – cualquiera funcionaría.
- Las **Variables de Tipo** denotan expresiones con tipo *desconocido*.
  - Identificadas con letras griegas minúsculas ($\alpha, \beta, \gamma, \ldots$).
  - A veces la variable terminará teniendo un tipo específico.
  - A veces la variable terminará indicando "cualquier tipo" – whatever...

UNIVERSIDAD SIMÓN BOLÍVAR

Variables de Tipo

Whatever floats your boat...

- En la última aplicación de `length` no solamente no está claro sino que *no importa* cuál es el contenido de la lista – cualquiera funcionaría.
- Las **Variables de Tipo** denotan expresiones con tipo *desconocido*.
  - Identificadas con letras griegas minúsculas ($\alpha, \beta, \gamma, \ldots$).
  - A veces la variable terminará teniendo un tipo específico.
  - A veces la variable terminará indicando "cualquier tipo" – whatever...
- Pueden cuantificarse universalmente
  - El tipo *más general* para `length` tendría una variable ligada

$$\forall\alpha.\mathit{list}(\alpha) \to \textbf{int}$$

  - Se puede *instanciar* la cuantificación con una variable libre particular

$$\mathit{list}(\alpha_1) \to \textbf{int}$$

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo aprovechar las Variables de Tipo?

Se “despejan”, pero se requiere consistencia

- Ayudan a determinar el uso consistente de un símbolo.

¿Cómo aprovechar las Variables de Tipo?

Se "despejan", pero se requiere consistencia

- Ayudan a determinar el uso consistente de un símbolo.
  - Aparece un símbolo `foo` sin declarar – su tipo será α.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo aprovechar las Variables de Tipo?

Se "despejan", pero se requiere consistencia

- Ayudan a determinar el uso consistente de un símbolo.
  - Aparece un símbolo `foo` sin declarar – su tipo será α.
  - Se usa `foo` como operando en una suma entera – hacemos $\alpha =$ **int**.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo aprovechar las Variables de Tipo?

Se "despejan", pero se requiere consistencia

- Ayudan a determinar el uso consistente de un símbolo.
  - Aparece un símbolo `foo` sin declarar – su tipo será α.
  - Se usa `foo` como operando en una suma entera – hacemos $\alpha =$ **int**.
  - Se encuentra *cos*(**foo**) – establecería α como **float**, pero ya está $\alpha =$ **int**, así que detectamos un error de tipos

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo aprovechar las Variables de Tipo?

Se "despejan", pero se requiere consistencia

- Ayudan a determinar el uso consistente de un símbolo.
  - Aparece un símbolo `foo` sin declarar – su tipo será α.
  - Se usa `foo` como operando en una suma entera – hacemos $\alpha =$ **int**.
  - Se encuentra *cos*(**foo**) – establecería α como **float**, pero ya está $\alpha =$ **int**, así que detectamos un error de tipos
- El proceso que permite determinar las valuaciones para variables de tipo a partir de la construcción del lenguaje en la cual se usan, se denomina **Inferencia de Tipos**.
- Lo frecuente es inferir el tipo de una función a partir de su *cuerpo*.

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo se usan?

```
type link : *cell;
procedure process (l : link, procedure p);
begin
  while l <> nil do begin
    p(l);
    l := *l.next
  end
end;
```

- process invoca a p sobre cada elemento de la lista l

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo se usan?

```
type link : *cell;
procedure process (l : link, procedure p);
begin
  while l <> nil do begin
    p(l);
    l := *l.next
  end
end;
```

- `process` invoca a `p` sobre cada elemento de la lista `l`
- El parámetro formal indica que `p` es un procedimiento – tipo $\alpha \to \beta$

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo se usan?

```
type link : *cell;
procedure process (l : link, procedure p);
begin
  while l <> nil do begin
    p(l);
    l := *l.next
  end
end;
```

- `process` invoca a `p` sobre cada elemento de la lista `l`
- El parámetro formal indica que `p` es un procedimiento – tipo $\alpha \to \beta$
- Se usa en una *instrucción*, no produce resultado – hacemos $\beta =$ **void**

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo se usan?

```
type link : *cell;
procedure process (l : link, procedure p);
begin
  while l <> nil do begin
    p(l);
    l := *l.next
  end
end;
```

- `process` invoca a `p` sobre cada elemento de la lista `l`
- El parámetro formal indica que `p` es un procedimiento – tipo $\alpha \to \beta$
- Se usa en una *instrucción*, no produce resultado – hacemos $\beta =$ **void**
- Recibe un argumento de tipo `link` – hacemos $\alpha =$ **link**.

Se infiere que todo uso de `process` debe recibir un
link $\to$ **void** como segundo argumento.

UNIVERSIDAD SIMÓN BOLÍVAR

Un lenguaje con funciones polimórficas

$P \rightarrow D \ ; \ E$
$D \rightarrow D \ ; \ D$
$D \rightarrow \textbf{id} : Q$
$Q \rightarrow \forall \ \textbf{type\_variable} \ . \ Q$
$Q \rightarrow T$
$T \rightarrow T \ -> \ T$
$T \rightarrow T \times T$
$T \rightarrow \textbf{unary\_constructor} \ (\ T \ )$
$T \rightarrow \textbf{basic\_type}$
$T \rightarrow \textbf{type\_variable}$
$T \rightarrow (\ T \ )$
$E \rightarrow E \ (\ E \ )$
$E \rightarrow E \ , \ E$
$E \rightarrow \textbf{id}$

- Funciones curryficadas o no.
- Tuplas.
- **unary_constructor** – para escribir *pointer*, *list*, ...
- **basic_type** – **void**, **bool**, **int**, **float**, ...

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo hacer la verificación?

Usaremos un Esquema de Traducción Dirigido por Sintaxis para etiquetar el árbol sintáctico – con tres observaciones importantes:

¿Cómo hacer la verificación?

Usaremos un Esquema de Traducción Dirigido por Sintaxis para etiquetar el árbol sintáctico – con tres observaciones importantes:

1. Cada ocurrencia de una función polimórfica *puede* tener argumentos de diferente tipo – las variables de tipos se *instancian* con un tipo particular a ese caso (¡qué podría tener variables!).

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo hacer la verificación?

Usaremos un Esquema de Traducción Dirigido por Sintaxis para etiquetar el árbol sintáctico – con tres observaciones importantes:

1. Cada ocurrencia de una función polimórfica *puede* tener argumentos de diferente tipo – las variables de tipos se *instancian* con un tipo particular a ese caso (¡qué podría tener variables!).
2. La presencia de variables obliga a operar usando equivalencia estructural – las variables se sustituyen por expresiones particulares para establecer una correspondencia (¡bidireccional!).

UNIVERSIDAD SIMÓN BOLÍVAR

¿Cómo hacer la verificación?

Usaremos un Esquema de Traducción Dirigido por Sintaxis para etiquetar el árbol sintáctico – con tres observaciones importantes:

1. Cada ocurrencia de una función polimórfica *puede* tener argumentos de diferente tipo – las variables de tipos se *instancian* con un tipo particular a ese caso (¡qué podría tener variables!).
2. La presencia de variables obliga a operar usando equivalencia estructural – las variables se sustituyen por expresiones particulares para establecer una correspondencia (¡bidireccional!).
3. Como una variable puede aparecer en varias expresiones, las instancias y sustituciones tienen que registrarse para asegurar su consistencia.

Un ejemplo informal antes de la teoría...

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa en el lenguaje

Varias definiciones seguidas de la expresión a verificar

Para el programa

$$\begin{aligned} \textbf{deref} &: \forall\alpha.\mathit{pointer}(\alpha) \to \alpha; \\ \textbf{q} &: \mathit{pointer}(\mathit{pointer}(\textbf{int})); \\ & \textbf{deref}(\textbf{deref}(\textbf{q})) \end{aligned}$$

UNIVERSIDAD SIMÓN BOLÍVAR

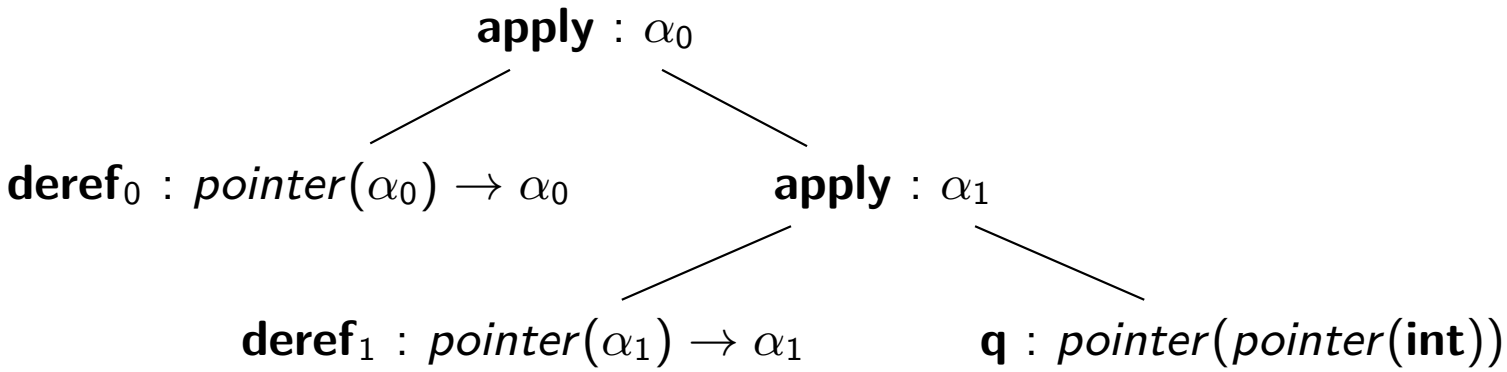
Un programa en el lenguaje

Varias definiciones seguidas de la expresión a verificar

Para el programa

$$\begin{aligned} \textbf{deref} &: \forall\alpha.\mathit{pointer}(\alpha) \to \alpha; \\ \textbf{q} &: \mathit{pointer}(\mathit{pointer}(\textbf{int})); \\ & \textbf{deref}(\textbf{deref}(\textbf{q})) \end{aligned}$$

se construye el árbol decorado



UNIVERSIDAD SIMÓN BOLÍVAR

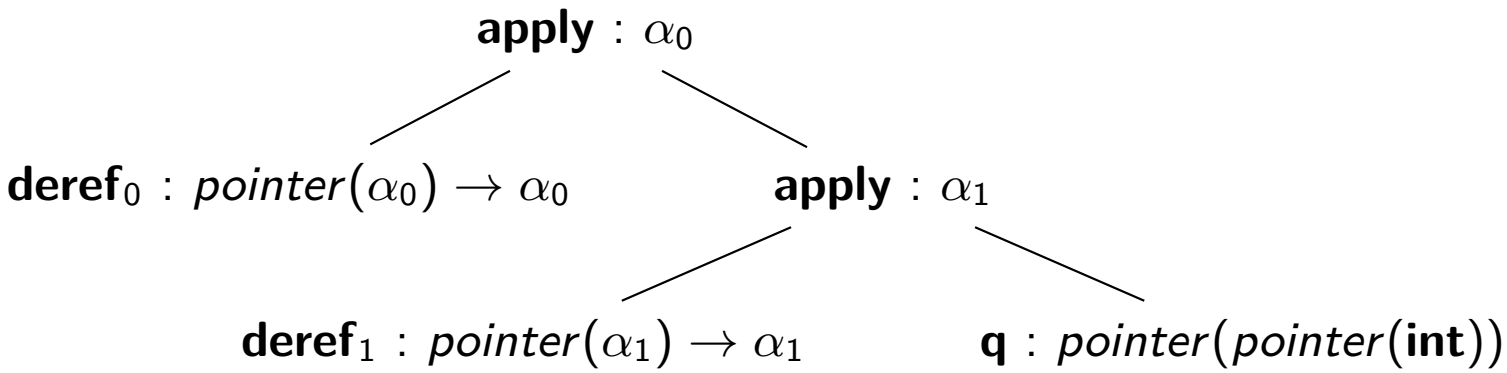
Un programa en el lenguaje

Varias definiciones seguidas de la expresión a verificar

Para el programa

$$\begin{aligned} \textbf{deref} &: \forall\alpha.\mathit{pointer}(\alpha) \to \alpha; \\ \textbf{q} &: \mathit{pointer}(\mathit{pointer}(\textbf{int})); \\ &\textbf{deref}(\textbf{deref}(\textbf{q})) \end{aligned}$$

se construye el árbol decorado



que permite inferir

$$\begin{aligned} \alpha_1 &= \mathit{pointer}(\textbf{int}) \\ \alpha_0 &= \textbf{int} \end{aligned}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Las variables y su reemplazo

Una **Sustitución** S define un conjunto de asociaciones entre variables de tipos y expresiones de tipos.

- $S(\alpha) = s_\alpha$ indica que en la sustitución S la variable de tipos α debe sustituirse por la expresión de tipos s_α.
- Si la sustitución S no especifica ninguna expresión para una variable particular α, se asume que $S(\alpha) = \alpha$.
- Si t es una expresión de tipos cualquiera, $S(t)$ es la expresión de tipos resultante de aplicar la sustitución S sobre t – se le denomina **instancia**.

Aplicar una sustitución es un procedimiento recursivo.

UNIVERSIDAD SIMÓN BOLÍVAR

Aplicación de una sustitución

function *subst*(t : *expr*) : *expr*

if t es un tipo primitivo **then**
return t
else if t es una variable **then**
return $S(t)$
else if $t = t_1 \to t_2$ **then**
return $S(t_1) \to S(t_2)$
end if

- *expr* corresponde a una expresión de tipos.
- Otros constructores de tipo – sustituir en cada argumento.
- Complejidad proporcional a la cantidad de nodos de la expresión.

UNIVERSIDAD SIMÓN BOLÍVAR

Algunos ejemplos triviales...

Siendo la sustitución

$$S = \{\alpha = \textbf{int}, \beta = \textbf{float}, \gamma = \mathit{pointer}(\beta)\}$$

Algunos ejemplos triviales...

Siendo la sustitución

$$S = \{\alpha = \textbf{int}, \beta = \textbf{float}, \gamma = \mathit{pointer}(\beta)\}$$

- $\mathit{pointer}(\textbf{int})$ es instancia de $\mathit{pointer}(\alpha)$.

UNIVERSIDAD SIMÓN BOLÍVAR

Algunos ejemplos triviales...

Siendo la sustitución

$$S = \{\alpha = \textbf{int}, \beta = \textbf{float}, \gamma = \mathit{pointer}(\beta)\}$$

- $\mathit{pointer}(\textbf{int})$ es instancia de $\mathit{pointer}(\alpha)$.
- $\mathit{pointer}(\textbf{float})$ es instancia de γ.

UNIVERSIDAD SIMÓN BOLÍVAR

Algunos ejemplos triviales...

Siendo la sustitución

$$S = \{\alpha = \textbf{int}, \beta = \textbf{float}, \gamma = \mathit{pointer}(\beta)\}$$

- $\mathit{pointer}(\textbf{int})$ es instancia de $\mathit{pointer}(\alpha)$.
- $\mathit{pointer}(\textbf{float})$ es instancia de γ.
- $\textbf{int} \to \textbf{int}$ es instancia de $\alpha \to \alpha$

UNIVERSIDAD SIMÓN BOLÍVAR

Algunos ejemplos triviales...

Siendo la sustitución

$$S = \{\alpha = \textbf{int}, \beta = \textbf{float}, \gamma = \mathit{pointer}(\beta)\}$$

- $\mathit{pointer}(\textbf{int})$ es instancia de $\mathit{pointer}(\alpha)$.
- $\mathit{pointer}(\textbf{float})$ es instancia de γ.
- $\textbf{int} \to \textbf{int}$ es instancia de $\alpha \to \alpha$
- $\textbf{int} \to \textbf{float}$ no es instancia de $\alpha \to \alpha$
  – reemplazo inconsistente de α

UNIVERSIDAD SIMÓN BOLÍVAR

Algunos ejemplos triviales...

Siendo la sustitución

$$S = \{\alpha = \textbf{int}, \beta = \textbf{float}, \gamma = \mathit{pointer}(\beta)\}$$

- $\mathit{pointer}(\textbf{int})$ es instancia de $\mathit{pointer}(\alpha)$.
- $\mathit{pointer}(\textbf{float})$ es instancia de γ.
- $\textbf{int} \to \textbf{int}$ es instancia de $\alpha \to \alpha$
- $\textbf{int} \to \textbf{float}$ no es instancia de $\alpha \to \alpha$
  – reemplazo inconsistente de α
- $\textbf{int} \to \alpha$ no es instancia de $\alpha \to \alpha$
  – ocurrencias de α sin reemplazar.

UNIVERSIDAD SIMÓN BOLÍVAR

Unificación

- Decimos que dos expresiones de tipos t_1 y t_2 se **unifican** si existe una sustitución S tal que $S(t_1) = S(t_2)$.
- En la práctica nos interesa la **unificación más general** S tal que
  - $S(t_1) = S(t_2)$ – es una sustitución.
  - $\forall S' \neq S : S'(t_1) = S'(t_2) \Rightarrow S'$ es instancia de S
  - S es el unificador que impone menos restricciones.
- De aquí en más al hablar de unificar está implícito que se trata del unificador más general.

Supongamos que sabemos encontrar el unificador más general.

UNIVERSIDAD SIMÓN BOLÍVAR

Esquema de Verificación

- Ante una definición – $\mathbf{id}_1(\mathbf{id}_2) = E$
  - Crear variables de tipo α y β.
  - Asociar el tipo $\alpha \to \beta$ con $\mathbf{id}_1$ – es la función.
  - Asociar el tipo α con $\mathbf{id}_2$ – es el argumento en el dominio.
  - Inferir un tipo para E – ha de ser $s \to t$.
  - Unificar y cuantificar variables que no tengan restricciones.

UNIVERSIDAD SIMÓN BOLÍVAR

Esquema de Verificación

- Ante una definición – $\mathbf{id}_1(\mathbf{id}_2) = E$
  - Crear variables de tipo α y β.
  - Asociar el tipo $\alpha \to \beta$ con $\mathbf{id}_1$ – es la función.
  - Asociar el tipo α con $\mathbf{id}_2$ – es el argumento en el dominio.
  - Inferir un tipo para E – ha de ser $s \to t$.
  - Unificar y cuantificar variables que no tengan restricciones.
- Ante una aplicación – $E_1(E_2)$
  - Inferir tipos para E_1 y E_2.
  - E_1 *debe* tener tipo de la forma $s \to s'$.
  - Sea t el tipo inferido para E_2.
  - Encontrar el unificador de s y t
    - Si existe, s' es el tipo inferido para $E_1(E_2)$.
    - Si no existe, hay un error de tipos

UNIVERSIDAD SIMÓN BOLÍVAR

Esquema de Verificación

- Ante una definición – $\mathbf{id}_1(\mathbf{id}_2) = E$
  - Crear variables de tipo α y β.
  - Asociar el tipo $\alpha \to \beta$ con $\mathbf{id}_1$ – es la función.
  - Asociar el tipo α con $\mathbf{id}_2$ – es el argumento en el dominio.
  - Inferir un tipo para E – ha de ser $s \to t$.
  - Unificar y cuantificar variables que no tengan restricciones.
- Ante una aplicación – $E_1(E_2)$
  - Inferir tipos para E_1 y E_2.
  - E_1 *debe* tener tipo de la forma $s \to s'$.
  - Sea t el tipo inferido para E_2.
  - Encontrar el unificador de s y t
    - Si existe, s' es el tipo inferido para $E_1(E_2)$.
    - Si no existe, hay un error de tipos
- Si ocurre una función polimórfica, eliminar el cuantificador con variables nuevas antes de inferir su tipo.

UNIVERSIDAD SIMÓN BOLÍVAR

Esquema de Verificación

- Ante una definición – $\mathbf{id}_1(\mathbf{id}_2) = E$
  - Crear variables de tipo α y β.
  - Asociar el tipo $\alpha \to \beta$ con $\mathbf{id}_1$ – es la función.
  - Asociar el tipo α con $\mathbf{id}_2$ – es el argumento en el dominio.
  - Inferir un tipo para E – ha de ser $s \to t$.
  - Unificar y cuantificar variables que no tengan restricciones.
- Ante una aplicación – $E_1(E_2)$
  - Inferir tipos para E_1 y E_2.
  - E_1 *debe* tener tipo de la forma $s \to s'$.
  - Sea t el tipo inferido para E_2.
  - Encontrar el unificador de s y t
    - Si existe, s' es el tipo inferido para $E_1(E_2)$.
    - Si no existe, hay un error de tipos
- Si ocurre una función polimórfica, eliminar el cuantificador con variables nuevas antes de inferir su tipo.
- Cada **id** encontrado por *primera* vez recibe una variable de tipos.

Verificando nuestro primer programa

$$\begin{aligned} \textbf{deref} &: \forall\alpha.\mathit{pointer}(\alpha) \to \alpha; \\ \textbf{q} &: \mathit{pointer}(\mathit{pointer}(\textbf{int})); \\ & \textbf{deref}(\textbf{deref}(\textbf{q})) \end{aligned}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Verificando nuestro primer programa

$$\textbf{deref} : \forall\alpha.\mathit{pointer}(\alpha) \rightarrow \alpha;$$
$$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}));$$
$$\textbf{deref}(\textbf{deref}(\textbf{q}))$$

Expresión : Tipo	Sustitución
$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}))$	

UNIVERSIDAD SIMÓN BOLÍVAR

Verificando nuestro primer programa

$$\textbf{deref} : \forall\alpha.\mathit{pointer}(\alpha) \to \alpha;$$
$$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}));$$
$$\textbf{deref}(\textbf{deref}(\textbf{q}))$$

Expresión : Tipo	Sustitución
$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}))$	
$\textbf{deref}_1 : \mathit{pointer}(\alpha_1) \to \alpha_1$	

UNIVERSIDAD SIMÓN BOLÍVAR

Verificando nuestro primer programa

$$\textbf{deref} : \forall\alpha.\mathit{pointer}(\alpha) \to \alpha;$$
$$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}));$$
$$\textbf{deref}(\textbf{deref}(\textbf{q}))$$

Expresión : Tipo	Sustitución
$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}))$	
$\textbf{deref}_1 : \mathit{pointer}(\alpha_1) \to \alpha_1$	
$\textbf{deref}_1(\textbf{q}) : \mathit{pointer}(\textbf{int})$	$\alpha_1 = \mathit{pointer}(\textbf{int})$

UNIVERSIDAD SIMÓN BOLÍVAR

Verificando nuestro primer programa

$$\textbf{deref} : \forall\alpha.\mathit{pointer}(\alpha) \to \alpha;$$
$$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}));$$
$$\textbf{deref}(\textbf{deref}(\textbf{q}))$$

Expresión : Tipo	Sustitución
$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}))$	
$\textbf{deref}_1 : \mathit{pointer}(\alpha_1) \to \alpha_1$	
$\textbf{deref}_1(\textbf{q}) : \mathit{pointer}(\textbf{int})$	$\alpha_1 = \mathit{pointer}(\textbf{int})$
$\textbf{deref}_0 : \mathit{pointer}(\alpha_0) \to \alpha_0$	

UNIVERSIDAD SIMÓN BOLÍVAR

Verificando nuestro primer programa

$$\textbf{deref} : \forall\alpha.\mathit{pointer}(\alpha) \to \alpha;$$
$$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}));$$
$$\textbf{deref}(\textbf{deref}(\textbf{q}))$$

Expresión : Tipo	Sustitución
$\textbf{q} : \mathit{pointer}(\mathit{pointer}(\textbf{int}))$	
$\textbf{deref}_1 : \mathit{pointer}(\alpha_1) \to \alpha_1$	
$\textbf{deref}_1(\textbf{q}) : \mathit{pointer}(\textbf{int})$	$\alpha_1 = \mathit{pointer}(\textbf{int})$
$\textbf{deref}_0 : \mathit{pointer}(\alpha_0) \to \alpha_0$	
$\textbf{deref}_0(\textbf{deref}_1(\textbf{q})) : \textbf{int}$	$\alpha_0 = \textbf{int}$

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

$$
\begin{aligned}
\textbf{length} &: \beta; \\
\textbf{x} &: \gamma; \\
\textbf{if} &: \forall\alpha.\textbf{bool} \times \alpha \times \alpha \to \alpha; \\
\textbf{null} &: \forall\alpha.\mathit{list}(\alpha) \to \textbf{bool}; \\
\textbf{tail} &: \forall\alpha.\mathit{list}(\alpha) \to \mathit{list}(\alpha); \\
\textbf{0} &: \textbf{int}; \\
\textbf{1} &: \textbf{int}; \\
+ &: \textbf{int} \times \textbf{int} \to \textbf{int}; \\
\textbf{match} &: \forall\alpha.\alpha \times \alpha \to \alpha; \\
&\quad \textbf{match}(\textbf{length}(\textbf{x}), \textbf{if}(\textbf{null}(\textbf{x}), \textbf{0}, \textbf{length}(\textbf{tail}(\textbf{x})) + \textbf{1}))
\end{aligned}
$$

La intención de **match** es establecer la equivalencia de los tipos de ambos argumentos – deducir el tipo de **length**.

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	

Un programa más interesante

Expresión : Tipo	Sustitución
x : γ	
length : β	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
x : γ	
length : β	
length(x) : δ	$\beta = \gamma \rightarrow \delta$

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
x : γ	
length : β	
length(x) : δ	$\beta = \gamma \to \delta$
x : γ	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \to \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \to \mathbf{bool}$	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \rightarrow \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \rightarrow \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
x : γ	
length : β	
length(x) : δ	$\beta = \gamma \rightarrow \delta$
x : γ	
null : $list(\alpha_n) \rightarrow$ **bool**	
null(x) : **bool**	$\gamma = list(\alpha_n)$
0 : **int**	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \rightarrow \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \rightarrow \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \rightarrow \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \rightarrow \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	
$\mathbf{tail} : \mathit{list}(\alpha_t) \rightarrow \mathit{list}(\alpha_t)$	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
x : γ	
length : β	
length(x) : δ	$\beta = \gamma \rightarrow \delta$
x : γ	
null : $list(\alpha_n) \rightarrow$ **bool**	
null(x) : **bool**	$\gamma = list(\alpha_n)$
0 : **int**	
x : $list(\alpha_n)$	
tail : $list(\alpha_t) \rightarrow list(\alpha_t)$	
tail(x) : $list(\alpha_n)$	$\alpha_t = \alpha_n$

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
x : γ	
length : β	
length(x) : δ	$\beta = \gamma \rightarrow \delta$
x : γ	
null : $list(\alpha_n) \rightarrow$ **bool**	
null(x) : **bool**	$\gamma = list(\alpha_n)$
0 : **int**	
x : $list(\alpha_n)$	
tail : $list(\alpha_t) \rightarrow list(\alpha_t)$	
tail(x) : $list(\alpha_n)$	$\alpha_t = \alpha_n$
length : $list(\alpha_n) \rightarrow \delta$	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \rightarrow \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \rightarrow \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	
$\mathbf{tail} : \mathit{list}(\alpha_t) \rightarrow \mathit{list}(\alpha_t)$	
$\mathbf{tail(x)} : \mathit{list}(\alpha_n)$	$\alpha_t = \alpha_n$
$\mathbf{length} : \mathit{list}(\alpha_n) \rightarrow \delta$	
$\mathbf{length(tail(x))} : \delta$	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
x : γ	
length : β	
length(x) : δ	$\beta = \gamma \rightarrow \delta$
x : γ	
null : $list(\alpha_n) \rightarrow$ **bool**	
null(x) : **bool**	$\gamma = list(\alpha_n)$
0 : **int**	
x : $list(\alpha_n)$	
tail : $list(\alpha_t) \rightarrow list(\alpha_t)$	
tail(x) : $list(\alpha_n)$	$\alpha_t = \alpha_n$
length : $list(\alpha_n) \rightarrow \delta$	
length(tail(x)) : δ	
1 : **int**	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \to \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \to \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	
$\mathbf{tail} : \mathit{list}(\alpha_t) \to \mathit{list}(\alpha_t)$	
$\mathbf{tail(x)} : \mathit{list}(\alpha_n)$	$\alpha_t = \alpha_n$
$\mathbf{length} : \mathit{list}(\alpha_n) \to \delta$	
$\mathbf{length(tail(x))} : \delta$	
$\mathbf{1} : \mathbf{int}$	
$+ : \mathbf{int} \times \mathbf{int} \to \mathbf{int}$	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \to \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \to \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	
$\mathbf{tail} : \mathit{list}(\alpha_t) \to \mathit{list}(\alpha_t)$	
$\mathbf{tail(x)} : \mathit{list}(\alpha_n)$	$\alpha_t = \alpha_n$
$\mathbf{length} : \mathit{list}(\alpha_n) \to \delta$	
$\mathbf{length(tail(x))} : \delta$	
$\mathbf{1} : \mathbf{int}$	
$+ : \mathbf{int} \times \mathbf{int} \to \mathbf{int}$	
$\mathbf{length(tail(x))} + 1 : \mathbf{int}$	$\delta = \mathbf{int}$

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \to \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \to \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	
$\mathbf{tail} : \mathit{list}(\alpha_t) \to \mathit{list}(\alpha_t)$	
$\mathbf{tail(x)} : \mathit{list}(\alpha_n)$	$\alpha_t = \alpha_n$
$\mathbf{length} : \mathit{list}(\alpha_n) \to \delta$	
$\mathbf{length(tail(x))} : \delta$	
$\mathbf{1} : \mathbf{int}$	
$+ : \mathbf{int} \times \mathbf{int} \to \mathbf{int}$	
$\mathbf{length(tail(x))} + 1 : \mathbf{int}$	$\delta = \mathbf{int}$
$\mathbf{if} : \mathbf{bool} \times \alpha_i \times \alpha_i \to \alpha_i$	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \to \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \to \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	
$\mathbf{tail} : \mathit{list}(\alpha_t) \to \mathit{list}(\alpha_t)$	
$\mathbf{tail(x)} : \mathit{list}(\alpha_n)$	$\alpha_t = \alpha_n$
$\mathbf{length} : \mathit{list}(\alpha_n) \to \delta$	
$\mathbf{length(tail(x))} : \delta$	
$\mathbf{1} : \mathbf{int}$	
$+ : \mathbf{int} \times \mathbf{int} \to \mathbf{int}$	
$\mathbf{length(tail(x))} + 1 : \mathbf{int}$	$\delta = \mathbf{int}$
$\mathbf{if} : \mathbf{bool} \times \alpha_i \times \alpha_i \to \alpha_i$	
$\mathbf{if(null(x), 0, length(tail(x))} + 1) : \mathbf{int}$	$\alpha_i = \mathbf{int}$

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \to \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \to \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	
$\mathbf{tail} : \mathit{list}(\alpha_t) \to \mathit{list}(\alpha_t)$	
$\mathbf{tail(x)} : \mathit{list}(\alpha_n)$	$\alpha_t = \alpha_n$
$\mathbf{length} : \mathit{list}(\alpha_n) \to \delta$	
$\mathbf{length(tail(x))} : \delta$	
$\mathbf{1} : \mathbf{int}$	
$+ : \mathbf{int} \times \mathbf{int} \to \mathbf{int}$	
$\mathbf{length(tail(x))} + 1 : \mathbf{int}$	$\delta = \mathbf{int}$
$\mathbf{if} : \mathbf{bool} \times \alpha_i \times \alpha_i \to \alpha_i$	
$\mathbf{if(null(x), 0, length(tail(x))} + 1) : \mathbf{int}$	$\alpha_i = \mathbf{int}$
$\mathbf{match} : \alpha_m \times \alpha_m \to \alpha_m$	

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Expresión : Tipo	Sustitución
$\mathbf{x} : \gamma$	
$\mathbf{length} : \beta$	
$\mathbf{length(x)} : \delta$	$\beta = \gamma \to \delta$
$\mathbf{x} : \gamma$	
$\mathbf{null} : \mathit{list}(\alpha_n) \to \mathbf{bool}$	
$\mathbf{null(x)} : \mathbf{bool}$	$\gamma = \mathit{list}(\alpha_n)$
$\mathbf{0} : \mathbf{int}$	
$\mathbf{x} : \mathit{list}(\alpha_n)$	
$\mathbf{tail} : \mathit{list}(\alpha_t) \to \mathit{list}(\alpha_t)$	
$\mathbf{tail(x)} : \mathit{list}(\alpha_n)$	$\alpha_t = \alpha_n$
$\mathbf{length} : \mathit{list}(\alpha_n) \to \delta$	
$\mathbf{length(tail(x))} : \delta$	
$\mathbf{1} : \mathbf{int}$	
$+ : \mathbf{int} \times \mathbf{int} \to \mathbf{int}$	
$\mathbf{length(tail(x))} + 1 : \mathbf{int}$	$\delta = \mathbf{int}$
$\mathbf{if} : \mathbf{bool} \times \alpha_i \times \alpha_i \to \alpha_i$	
$\mathbf{if(null(x), 0, length(tail(x))} + 1) : \mathbf{int}$	$\alpha_i = \mathbf{int}$
$\mathbf{match} : \alpha_m \times \alpha_m \to \alpha_m$	
$\mathbf{match(length(x), if(null(x), 0, length(tail(x))} + 1)) : \mathbf{int}$	$\alpha_m = \mathbf{int}$

UNIVERSIDAD SIMÓN BOLÍVAR

Un programa más interesante

Finalmente...

El resultado del algoritmo es la sustitución

$$S = \left\{ \begin{array}{c} \alpha_i = \textbf{int} \\ \alpha_m = \textbf{int} \\ \delta = \textbf{int} \\ \gamma = \mathit{list}(\alpha_n) \\ \alpha_t = \alpha_n \\ \beta = \gamma \to \delta \end{array} \right\}$$

como no se hace ninguna suposición sobre α_n, podemos cuantificar y obtener el tipo inferido

$$\textbf{length} : \forall \alpha_n . \mathit{list}(\alpha_n) \to \textbf{int}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Deja Vu

¡Sustituir y unificar es lo que hacemos en el sistema formal!

- La regla para inferir una aplicación funcional sería

$$\frac{\rho \vdash e_1 : \alpha \to \beta \qquad \rho \vdash e_2 : \alpha}{\rho \vdash e_1 \ (\ e_2 \ ) : \beta}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Deja Vu

¡Sustituir y unificar es lo que hacemos en el sistema formal!

- La regla para inferir una aplicación funcional sería

$$\frac{\rho \vdash e_1 : \alpha \to \beta \qquad \rho \vdash e_2 : \alpha}{\rho \vdash e_1 \ (\ e_2 \ ) : \beta}$$

- Y puedo inferir el tipo para una llamada

$$\frac{\rho \vdash \mathit{length} : \forall \alpha.\mathit{list}(\alpha) \to \mathbf{int} \qquad \dfrac{\ldots}{\rho \vdash [42, 17, 69] : \mathit{list}(\mathbf{int})}}{\rho \vdash \mathit{length} \ (\ [42, 69, 17] \ ) : \mathbf{int}}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Unificación

- ¿Cómo encontrar (si existe) una sustitución S que permita hacer idénticas las expresiones s y t?
- El caso trivial de unificación es la comparación de igualdad – si s es idéntica a t, la sustitución vacía les unifica.
- Para generalizar a expresiones de tipos, debemos considerar la comparación estructural entre grafos incluyendo ciclos.
  - Las variables de tipos estarán en las hojas – *singleton*.
  - Los tipos primitivos estarán en las hojas – *singleton*.
  - Los constructores de tipos estarán en los nodos interiores.
- Construiremos clases de equivalencia para los nodos.

UNIVERSIDAD SIMÓN BOLÍVAR

Un ejemplo

Consideremos las expresiones

$$((\alpha_1 \to \alpha_2) \times \mathit{list}(\alpha_3)) \to \mathit{list}(\alpha_2)$$
$$((\alpha_3 \to \alpha_4) \times \mathit{list}(\alpha_3)) \to \alpha_5$$

UNIVERSIDAD SIMÓN BOLÍVAR

Un ejemplo

Consideremos las expresiones

$$((\alpha_1 \to \alpha_2) \times \mathit{list}(\alpha_3)) \to \mathit{list}(\alpha_2)$$
$$((\alpha_3 \to \alpha_4) \times \mathit{list}(\alpha_3)) \to \alpha_5$$

Pueden unificarse con la sustitución

$$S = \left\{ \begin{array}{l} \alpha_1 = \alpha_1 \\ \alpha_2 = \alpha_2 \\ \alpha_3 = \alpha_1 \\ \alpha_4 = \alpha_2 \\ \alpha_5 = \mathit{list}(\alpha_2) \end{array} \right\}$$

UNIVERSIDAD SIMÓN BOLÍVAR

Un ejemplo

Consideremos las expresiones

$$((\alpha_1 \to \alpha_2) \times \mathit{list}(\alpha_3)) \to \mathit{list}(\alpha_2)$$
$$((\alpha_3 \to \alpha_4) \times \mathit{list}(\alpha_3)) \to \alpha_5$$

Pueden unificarse con la sustitución

$$S = \left\{ \begin{array}{l} \alpha_1 = \alpha_1 \\ \alpha_2 = \alpha_2 \\ \alpha_3 = \alpha_1 \\ \alpha_4 = \alpha_2 \\ \alpha_5 = \mathit{list}(\alpha_2) \end{array} \right\}$$

obteniendo

$$((\alpha_1 \to \alpha_2) \times \mathit{list}(\alpha_1)) \to \mathit{list}(\alpha_2)$$

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Unificación

Infraestructura

- *Node* – registro con campos para un operador binario, hijos izquierdo y derecho, y apuntador a la clase de equivalencia.
  - Cada clase de equivalencia tiene un nodo representante – se identifica porque tiene el apuntador nulo.
  - Resto de los nodos en la clase de equivalencia apuntan a su representante – directa o indirectamente.
  - Inicialmente, todos los nodos están en su propia clase de equivalencia.
- *find*(*node*) – encuentra el nodo representante para la clase de equivalencia a la cual pertenece *node*.
- *union*(m, n) – combina las clases de equivalencia de m y n.
  - Si uno de los dos **no** es una variable, ese será el nuevo representante.
  - Basta hacer que el representante de uno apunte al representante del otro para completar la combinación.

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Unificación

El algoritmo

function $unify(m, n : Node)$: **bool**

$s \leftarrow find(m)$
$t \leftarrow find(n)$
if $s = t$ **then**
  return true
else if s y t son el mismo tipo básico **then**
  return true
else if $s = op(s_1, s_2)$ **and** $t = op(t_1, t_2)$ **then**
  $union(s, t)$
  return $unify(s_1, t_1)$ **and** $unify(s_2, t_2)$
else if $var(s)$ **or** $var(t)$ **then**
  $union(s, t)$
  return true
else
  return false
end if

UNIVERSIDAD SIMÓN BOLÍVAR

Algoritmo de Unificación

¿Cómo encuentro la sustitución?

- Supongamos que m o n es una hoja – corresponde a una variable.
- Supongamos que ha sido incluida en una clase de equivalencia con un nodo que representa una expresión.
- Entonces, para cada variable α
  - $find(\alpha)$ obtiene el nodo representante r para su clase de equivalencia.
  - La expresión representada por r corresponde a $S(\alpha)$.

UNIVERSIDAD SIMÓN BOLÍVAR

Bibliografía

- [*Aho*] (Primera Edición)
  - Capítulo 6.6
  - Ejercicios 6.20 a 6.31
- [*Aho*] (Segunda Edición)
  - Secciones 6.3.1, 6.3.2, 6.5.1, 6.5.2 y 6.5.3
  - Ejercicios 6.5.1 y 6.5.2

UNIVERSIDAD SIMÓN BOLÍVAR