# Compiladores: Sesión 20. Análisis semántico, verificación e inferencia de tipos

Prof. Gloria Inés Alvarez V.

Departamento de Ciencias e Ingeniería de la Computación
Pontificia Universidad Javeriana Cali

3 de abril de 2008

# Funciones Polimórficas

- Una función polimórfica es aquella que ejecuta las mismas instrucciones sobre argumentos de diferentes tipos.
- La existencia de polimorfismo debe ser tenida en cuenta por un sistema de verificación de tipos, ello requiere extender las expresiones de tipo adicionando variables de tipo.
- La existencia de variables de tipo hace más complejo el establecer la equivalencia de dos expresiones de tipo.

# Variables de Tipo

Permiten referirse a un tipo desconocido dentro de una expresión de tipo, se denotan con letras griegas. Se usan para:

- Revisar el uso consistente de identificadores en lenguajes que no requieren que ellos sean declarados previamente.
- Asignar tipo a expresiones relacionadas con funciones polimórficas.

# Inferencia de Tipos

**Definición**

*Es el problema de determinar el tipo de una construcción del lenguaje a partir de la forma en que es usado.*

Las técnicas para hacer inferencia y verificación son similares, en ambos casos hay que tratar con variables de tipo.

# Reglas de Tipo para Funciones Polimórficas

1. Diferentes ocurrencias del mismo nombre de función en la misma expresión de tipo, pueden tener argumentos de diferente tipo.
2. Al existir variables de tipo es necesario unificar las expresiones relacionadas. La unificación determina si dos expresiones de tipo $s$ y $t$ son estructuralmente equivalentes reemplazando las variables de tipo por expresiones de tipo. Ejemplo: $pointer(\alpha) = pointer(pointer(integer)) \Rightarrow \alpha = pointer(integer)$
3. Se necesita un mecanismo para recordar el efecto de la unificación en las dos expresiones. Si la unificación de $S$ y $S'$ resulta en una variable $\alpha$ representando el tipo $t$, entonces $\alpha$ tiene que continuar representando el tipo $t$ a medida que avanza la verificación.

# Sustituciones, Instancias y Unificación

**Definición**

*Una sustitución es un mapeo de variables de tipo a expresiones de tipo.*

**Definición**

*Una instancia es una expresión de tipo a la cual se le ha aplicado una sustitución*

# Algoritmo para aplicar una sustitución

sustituir(t: expresionTipo): expresionTipo
{
    if t es un tipo básico then
        return t
    if t es una variable then
        return S(t)
    if t es $t_1 \longrightarrow t_2$ then
        return sustituir($t_1$) $\longrightarrow$ sustituir($t_2$)
}

# Sustituciones

- S(t) es la expresión de tipo que resulta cuando se aplica sustituir a t.
- S(t) es una instancia de t.
- Si la sustitución no especificia una expresión para la variable $\alpha$, se entiende que S($\alpha$) = $\alpha$.
- Ejemplo

# Instancias

- Notación: $s < t$ significa que s es instancia de t
- Ejemplos donde s es instancia de t
  - pointer(integer) $<$ pointer($\alpha$)
  - pointer($\alpha$) $< \beta$
- Ejemplos donde s no es instancia de t
  - integer $\not<$ real
  - integer $\rightarrow$ real $\not< \alpha \rightarrow \alpha$
  - integer $\rightarrow \alpha \not< \alpha \rightarrow \alpha$

# Unificación

## Definición

*Dos expresiones de tipo $t_1$ y $t_2$ unifican, si existe una sustitución $S$ tal que $S(t_1) = S(t_2)$.*

En la práctica se busca encontrar el unificador más general, es decir, la sustitución que establece el menor número de restricciones posible sobre las variables de las expresiones.

## Definición

*El unificador más general de $t_1$ y $t_2$ es una sustitución que cumple:*

- $S(t_1) = S(t_2)$
- $\forall S'$ *tal que* $S'(t_1) = S'(t_2)$ *se tiene que* $S'$ *es una instancia de* $S$.

# Verificación de tipo de funciones polimórficas

- Se parte de una representación en grafos de las expresiones de tipo
- Las reglas para verificar expresiones de tipo se escriben en términos de las siguientes operaciones:
  - fresh(t) reemplaza las variables acotadas en t por variables libres, al hacerlo elimina los cuantificadores universales
  - unificar(m,n) unifica las expresiones de tipo, como efecto de borde, mantiene un registro de la sustitución que se hace en la expresión equivalente. Si la unificación falla, también falla la verificación de tipos.

# Verificación de tipo de funciones polimórficas

$E \rightarrow E_1(E_2)$ {p = mkleaf(newtypevar); unify($E_1$.type, mknode($\rightarrow$,$E_2$.type,p)); E.type=p}
$E \rightarrow E_1, E_2$ {E.type = mknode($\times$,$E_1$.type,$E_2$.type)}
$E \rightarrow id$ {E.type = fresh(id.type)}

# Ejemplo de verificación de tipos

Suponer las siguientes expresiones de tipo: deref:
$\forall\alpha.\mathit{pointer}(\alpha \rightarrow \alpha)$
q: pointer(pointer(integer))
Y suponer la expresión: deref(deref(q))

# Ejemplo representación en grafo [Aho 2a. ed. p. 398]

Sean las expresiones de tipo: $((\alpha_1 \rightarrow \alpha 2) \times \mathit{list}(\alpha_3)) \rightarrow \mathit{list}(\alpha_2)$ y $((\alpha_1 \rightarrow \alpha 2) \times \mathit{list}(\alpha_3)) \rightarrow \alpha_5$

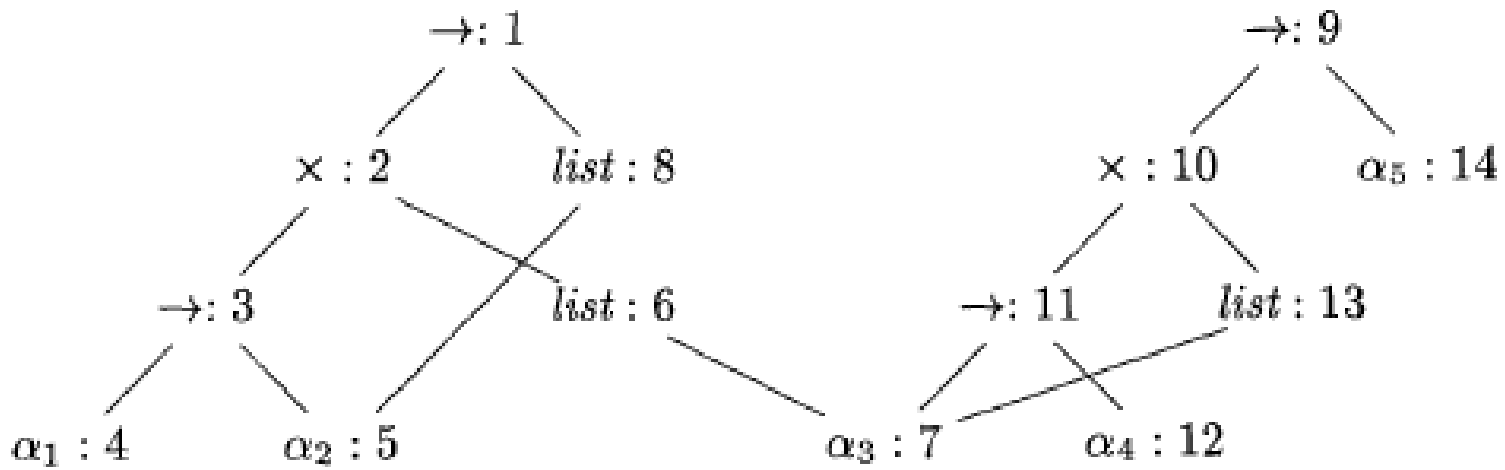


Figure 6.33: Initial graph with each node in its own equivalence class

# Algoritmo de Unificación [Aho 2a. ed. p. 397]

```
boolean unify(Node m, Node n) {
        s = find(m); t = find(n);
        if ( s = t ) return true;
        else if ( nodes s and t represent the same basic type ) return true;
        else if (s is an op-node with children s1 and s2 and
                        t is an op-node with children t1 and t2) {
                union(s, t);
                return unify(s1, t1) and unify(s2, t2);
        }
        else if s or t represents a variable {
                union(s, t);
                return true;
        }
        else return false;
}
```

# Resultado de la unificación [Aho 2a. ed. p. 396]

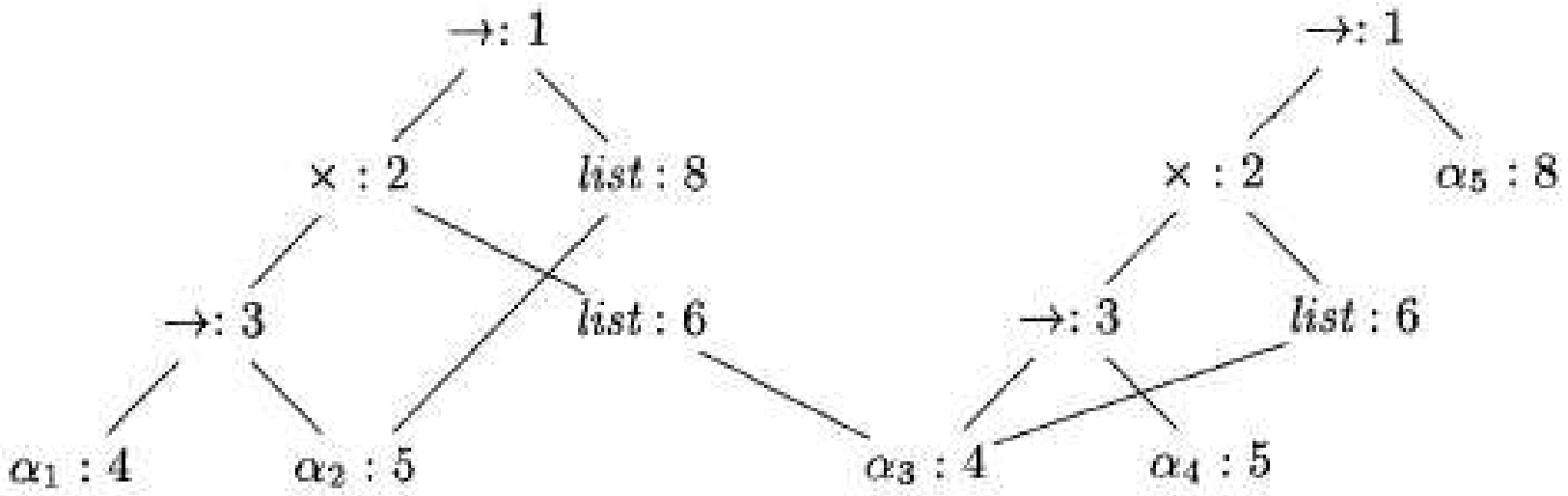


Figure 6.31: Equivalence classes after unification