Primera práctica de lenguajes de programación
Algoritmo de unificación
Implementación en C++

Edwin Andrey Duque Loaiza
Andrés Sicard Ramírez
Edison Valencia Díaz
Juan Francisco Cardona Mc'Cormick

26 de Septiembre de 2019

1. Introducción

En la verificación o inferencia de tipos en los lenguajes de programación, se requiere comprobar que dos tipos son equivalentes. La noción de equivalencia dice que dos tipos son equivalentes si se encuentra un sustitución que transforme ambos tipos en el mismo. Por ejemplo, si tenemos el tipo de dato $\tau_1 = \text{Int}$ y el tipo de dato $\tau_2 = \text{Int}$, la equivalencia entre ambos es obvia:

$$\text{Int} \equiv \text{Int}$$

Puesto ambos son iguales su sustitución θ es vacía.

En otros casos, como: $\tau_3 = a \to a$ y $\tau_4 = \text{Int} \to \text{Int}$, se requiere de un poco más de esfuerzo

$$a \to a \equiv \text{Int} \to \text{Int}$$

En esta ecuación se debe sustituir uno de los lados de la ecuación para obtener algo que sea equivalente, en este caso la sustitución obvia es sustituir a la variable a por el tipo Int y aplicar dicha sustitución ($\theta = a \mapsto Int$) a la ecuación, lo que nos produce:

$$Int \to Int \equiv Int \to Int$$

Robinson [5] definió un algoritmo llamado de unificación que permite solucionar este tipo de problemas. En los cuales recibe una seríe de restricciones de la forma $T \equiv S$ y encuentra un conjunto de sustituciones que permite transformar un tipo en el otro. El algoritmo de unificación para los tipos de datos sirve para "igualar" dos tipos (ver 3).

2. Algoritmo

En algoritmo 1 es una copia del encontrado en el libro de Pierce [3].

Algorithm 1: $unify(C)$

```
   Data: C = {(S ≡ T) | S ∈ Type ∧ T ∈ Type}
   Result: Θ = {(X ↦ T) | X ∈ Var ∧ T ∈ Type}
1  begin
2      if C = ∅ then
3          [ ]
4      else
5          let {S ≡ T} ∪ C' = C in if S = T then
6              unify(C');
7          else if S = X and X ∉ FV(T) then
8              unify([X ↦ T]C') ∘ [X ↦ T];
9          else if T = X and X ∉ FV(T) then
10             unify([X ↦ S]C') ∘ [X ↦ S];
11         else if S ≡ S1 → S2 and T ≡ T1 → T2 then
12             unify(C' ∪ {S1 ≡ T1, S2 ≡ T2});
13         else
14             fail;
```

El algoritmo toma como entrada un conjunto de restricciones C, donde cada restricción es un par ordenado de forma $\{S = T\}$ y cada elemento del par ordenado es un tipo (ver 3). El algoritmo retorna un conjunto de sustituciones Θ y cada sustitución es de la forma $\{X \mapsto T\}$, donde $X \in Var$ es una variable y T es un tipo (ver 3).

En la línea 2, el algoritmo verifica si el conjunto de restriciones C es vacío retorna un conjunto Θ también vacío.

En la línea 5, el algoritmo separa el conjunto restriciones C en dos partes: la primera es el primer elemento del conjunto $\{S \equiv T\}$ y la segunda parte es el resto del conjunto C'. También en ésta línea se verifica si los tipos básicos son idénticos, si este es el caso realiza la unificación al resto del conjunto C'.

En la línea 7, el algoritmo verifica si uno de los tipos es una variable X y ella no pertence al conjunto de variables libres $FV(S)$ (ver 5.1) del otro tipo S. Si este es el caso, en la línea 8 realiza la unificación con la aplicación (ver 5.2) de la sustitución $\theta = X \mapsto S$ al conjunto C'; el resultado de la unificación es compuesto $\circ$ (ver 5.3) con la sustitución $\theta = X \mapsto S$.

En la línea 9, el algoritmo verifica si uno de los tipos es una variable X y ella no pertence al conjunto de variables libres $FV(T)$ (ver 5.1) del otro tipo t. Si este es el caso, en la línea 8 realiza la unificación con la aplicación (ver 5.2) de la sustitución $\theta = X \mapsto t$ al conjunto C'; el resultado de la unificación es

compuesto $\circ$ (ver 5.3) con la sustitución $\theta = X \mapsto T$.

En la línea 11, el algoritmo verifica si ambos tipos son funciones: $\{S \equiv S_1 \to S_2\}$ y $\{T \equiv T_1 \to T_2\}$. Realiza la unificación con el conjunto C' unido con dos restricciones más: el tipo $S_1 \equiv T_1$ y $S_2 \equiv T_2$.

En la línea 14, el algoritmo llega si ninguna de la anteriores condiciones se cumple, por lo tanto no puede unificar y falla.

3. Lenguajes de tipos

Uno de los elementos que requiere el algoritmo de unificación son los tipos de datos τ:

$$\begin{array}{rcl} \tau & \to_1 & X \in Var \\ & |_2 & \text{Int} \\ & |_3 & \text{Bool} \\ & |_4 & \tau_1 \to \tau_2 \end{array}$$

En la primera producción un tipo es una variable X que pertenece al conjunto de variables Var (ver 4.1). La segunda producción identifica que el tipo es un entero Int. La tercera producción indica que es un tipo booleano Bool. La cuarta y última, indica que el tipo es una función que recibe un tipo τ_1 y retorna un tipo τ_2.

3.1. Ejemplos de tipos

Los tipos básicos son: enteros Int, booleanos Bool y variables de tipos Var. Ejemplos de los tipos simples:

Int	Un entero
Bool	Un booleano
a	Una variable
bcd1	Una variable

Los tipos funciones permite construir tipos a partir de los más sencillos:

$\text{Int} \to \text{Int}$	Función que toma un entero y retorna un entero
$\text{Int} \to \text{Int} \to \text{Int}$	Un operador binario: $+ \ - \ \times$
$\text{Int} \to \text{Bool}$	Toma un entero y verifica una propiedad en él
$\text{a} \to \text{Int}$	Una función polimorfica que convierte a entero
$\text{a} \to \text{a} \to \text{a}$	Un operador binario genérico

4. Estructura de datos

A continuación se muestra una descripción de los tipos de datos utilizados por la función $unify$ y sus funciones auxiliares (ver 5).

4.1. Variables

La variables son cualquier cadena de carácteres que comienza con una letra, seguida de una secuencia posiblemente vacía de letras y números. Su expresión regular:

$$[a..zA..Z]([a..zA..Z] \mid [0..\ 9])^*$$

4.2. Tipos

Los tipos definidos en 3 deben ser definidos como una estructura de datos. Esta estructura debe ser capaz de representar cualquiera de los cuatro elementos que consta un tipo τ.

4.3. Restricción

Una restricción es un par ordenado de tipos (S, T) donde $T \in \tau$ y $S \in \tau$.

4.4. Sustitución

Una sustitución es un par ordenado de tipos (X, T) donde $X \in Var$ y $T \in \tau$. O también la sustitución puede ser definida como una función:

$$\theta :: \tau -> \tau.$$

Es una función que recibe un tipo y lo tranforma en otro tipo.

4.5. Conjuntos

Existen dos tipo de conjuntos: uno de restricciones C y otro de sustituciones Θ.

5. Funciones auxiliares

Existe una serie de funciones auxilares que son necesarias que el algoritmo de $unify$.

5.1. Variables libres, *Free Variables* (FV)

La función FV recibe un tipo de dato y retorna un conjunto de variables que son están definidas dentro del tipo:

$$FV(T) = \begin{cases} \{X\}, & \text{si } T = X \text{ donde } X \text{ es una variable} \\ \emptyset & \text{si } T = \text{Int} \\ \emptyset & \text{si } T = \text{Int} \\ FV(\tau_1) \cup FV(\tau_2) & \text{si } T = \tau_1 \to \tau_2 \end{cases}$$

5.2. Aplicación de la sustitución

La aplicación α de una sustitición es una operación que está divida en varia sub-operaciones: en primer lugar la aplicación de la sustitución en un conjunto de restricciones:

$$\alpha :: \theta \to C \to C$$

Toma una sustitución se la aplica a cada elemento del conjunto de restricciones y produce un nuevo conjunto de restricciones. Esta sustitución α está definida de la siguiente forma:

$$\alpha = \begin{cases} \emptyset & C = \emptyset \\ \{\alpha\ \theta\ c\} \cup\ \alpha\ \theta\ C' & C = \{c\} \cup C' \end{cases}$$

Esta aplicación está basada en otra aplicación de sustitución sobre una restricción $\rho = \{S \equiv T\}$, donde $S \in \tau$ y $T \in \tau$:

$$\alpha :: \theta \to \rho \to \rho$$

y está definida como:

$$\alpha = \{(\alpha\ \theta\ S) \equiv (\alpha\ \theta\ T)\}$$

Esta aplicación también está basada en otra aplicación α pero sobre tipos sobre los tipos:

$$\alpha = \theta \to \tau \to \tau$$

Si la sustitución θ es definido como $[X' \mapsto S']$, entonces la aplicación α está definida como:

$$\alpha\ \theta\ \tau = \begin{cases} X & \text{si } \tau = X \text{ y } X \neq X' \\ S' & \text{si } \tau = X \text{ y } X = X' \\ \text{Int} & \text{si } \tau = \text{Int} \\ \text{Bool} & \text{si } \tau = \text{Bool} \\ (\alpha\ \theta\ \tau_1)\ \to (\alpha\ \theta\ \tau_2) & \text{si } \tau = \tau_1 \to \tau_2 \end{cases}$$

5.3. Composición de sustituciones

La composición ($\circ$) de sustituciones se puede definir de dos formas. Primero, si la sustitución está definida como un par ordenado la composición es una función que toma un conjunto de sustituciones Θ y una sustitución θ y adiciona está última al conjunto de sustituciones.

Segundo, si la sustitucion está definida como una función, se puede utilizando el operador de composición de funciones, que toma dos sustituciones y produce una nueva.

6. Gramática de entrada

La siguiente es la definición de la gramática de entrada:

$$\begin{array}{rcl} Entrada & \rightarrow & ListaEquivEOF \\ ListaEquiv & \rightarrow & \tau = \tau\ EOL\ ListaEquiv \\ & \rightarrow & \epsilon \\ \tau & \rightarrow & VAR \\ & \rightarrow & int \\ & \rightarrow & bool \\ & \rightarrow & \tau \rightarrow \tau \\ & \rightarrow & (\tau) \end{array}$$

Donde `EOF` es final de fichero, `EOL` es el carácter de línea nueva[1].
Ejemplo:
De una entrada correcta

```
a = a <EOL>
int = int <EOL>
a -> a = int -> int <EOL>
(a -> a) -> b = (int -> int) -> int <EOL>
<EOF>
```

La entrada muestra un programa de 4 líneas de tipos que será verificados si son equivalentes o no. Los símbolos `<EOL>` y `<EOF>` se refieren a la línea nueva y al final del fichero, el primer caracter es visible y el segundo es simple un evento.

7. Requerimientos

7.1. Técnicos

Lenguajes de programación: C++

Nombre ejecutable: cppunify

Control de versiones: Se va utilizar el manejador de versiones *subversion* [4].

Repositorio de control de versiones: Riouxsvn (`http://www.riouxsvn.com`).

Estructura de directorios: La siguiente la estructura de directorios o carpetas o *folders* que tendrá el repositorio, donde `+` es un directorio y `-` es un archivo.

`miembros.xml` archivo que contiene la definición de los miembros del grupo. El siguiente es el contenido de un posible grupo.

[1] En la plataforma cygwin-Windows equivale al caracter cuyo valor numérico es 10.

```
<grupo>
   <nombre>UnificadorSupremo</nombre>
   <url>https://subversion.assembla.com/svn/unificador/</url>
   <miembro>
      <nombre>Juan Francisco Cardona McCormick</nombre>
      <codigo>198610250010</codigo>
      <email>fcardona@eafit.edu.co</email>
   </miembro>
   <miembro>
      <nombre>Alejandro Gómez Londoño</nombre>
      <codigo>201010001010</codigo>
      <email>alegomez544@eafit.edu.co</email>
   </miembro>
</grupo>
```

El *url* identifica el repositorio donde está almacenada la práctica. El nombre del grupo es de una sola palabra, en minúsculas. Se debe añadir el correo electrónico en la segunda entrega. *mainclases* es una lista con las clases principales del proyecto tanto para ANTRL como para CUP.

El <NombreGrupo> es el nombre elegido para su grupo un identificador que sigue el mismo formato de los identificadores en Java.

El fichero`README` contiene la información del contenido de las sub-capertas y como compilar cada proyecto.

Los directorios `src-[leng]`$_i$ son los directorios que contienen los fuentes de cada uno de los lenguajes que han hecho las versiones de la práctica:

$$[\text{leng}] \rightarrow \text{cplusplus} \mid \text{csharp} \mid \text{clisp} \mid \text{haskell}$$

El directorio `examples` contiene los diferentes ejemplos con los cuales corren la práctica.

7.2. Administrativos

Grupo de trabajo: Grupo máximo de dos estudiantes. *Cada* miembro del equipo debe registrarse en assembla (`http://www.assembla.com`) con un usuario propio. Uno de los miembros debe crear un repositorio donde los miembros del equipo son los dueños del repositorio. Deben enviar una invitación al monitor de la materia y al profesor de la misma.

Primera entrega: Viernes 26 de Octubre a las 12:00 m. Esta se hará automática a través del sistema de control de versiones. Se *debe* cumplir todos los requerimientos señalados en este documento. Recuerde que la idea es entregar una gramática en el archivo de las gramáticas.

Control de versiones: El control de versiones no es solamente un herramienta que facilite la comunicación entre los miembros del grupo y del control de versiones, sino que también ayudará al profesor a llevar un control sobre el desarrollo de la práctica. Se espera que las diferentes registros

dentro del control de versiones sean cambios graduales. En caso contrario, se procederá a realizar un escutrinio a fondo del manejo de control de versiones para evitar fraudes.

Bibliografía

[1] Arnold, Ken. Gosling, James. *The Java Programming Language*. 2005.

[2] Hindley, J. Roger. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Societe*, 146:29-60. 1969.

[3] Pierce, Benjamin C. *Type and Programming Languages*. The MIT Press. 2002.

[4] Pilato, Michael C. Collins-Sussman, Ben. Fitzpatrick, Brian W. *Version Control with Subversion*. 2008.

[5] Robinson, J. Alan. Computational logic: The unification computation. *Machine Intelligence*, 6:63-72, 1971.