Viper

Análisis Semántico

Compiladores
Universidad Galileo

1 Reglas de Tipos

Esta sección define formalmente las *reglas de tipos* del lenguaje **Viper**. Las reglas de tipos definen los tipos de cada expresión del lenguaje dado un contexto. El contexto es el *entorno de tipos*, que describe el tipo de cada identificador que aparece en una expresión. El entorno de tipos es descrito en la Sección 1.1. La Sección 1.2 definen las reglas de inferencia.

1.1 Entornos de Tipos

La razón de tener entornos de tipos es debido al problema que se encuentra en el caso de que una expresión v sea un identificador. No es posible decir que tipo tiene v, necesitamos saber el tipo con el que fue declarado v. Esa declaración tiene que existir para cada identificador local en un programa válido de Viper.

Para capturar la información acerca de los tipos de los identificadores, nosotros usamos un *entorno de tipos*. El entorno consiste de 2 partes: un entorno de funciones M y un entorno de identificadores O. El entorno de funciones y el de identificadores son funciones (también llamados *mappings*). El entorno de identificadores tiene la siguiente forma:

$$O(v) = T$$

que asigna el tipo T al identificador v. El entorno de funciones es más complejo, es una función de la forma:

$$M(f) = (T_1, \ldots, T_{n-1}, T_n)$$

donde f es el nombre de la función y $T_1, \ldots, T_n$ son tipos. La tupla de tipos es la firma de la función. La interpretación de las firmas es que la función f tiene parámetros definidos de tipo $(T_1, \ldots, T_{n-1})$ (en ese orden) y el tipo de retorno es T_n. Dos mappings son requeridos en vez de uno porque los nombres de los identificadores y funciones no causan conflictos, es decir, puede haber una función y un identificador con el mismo nombre.

A cada expresión e se le verifica su tipo en un entorno de tipos, cada subexpresión de e se le verifica su tipo en el mismo entorno de tipos o, si se introduce un nuevo identificador, en un entorno modificado.
Consideren lo siguiente:

```
int c = 33;
...
```

La declaración local de un identificador (que no es una expresión), introduce una nueva variable `c` de tipo `int`. Si O es el entorno de identificadores donde apareció la declaración, entonces lo que sigue después es verificado en el entorno de tipos:

$$O[int/c]$$

donde la notación $O[T/c]$ es definida de la siguiente manera:

$$O[T/c](c) = T$$

$$O[T/c](d) = O(d) \quad si \quad d \neq c$$

1.2 Reglas de Verificación de Tipos

La forma general de una regla de inferencia es:

$$\frac{\vdots}{O, M \vdash e : T}$$

La regla de inferencia se debería de leer: En el entorno de identificadores O y funciones M, la expresión e tiene tipo T. Los puntos arriba de la barra horizontal es donde van las declaraciones acerca de las subexpresiones de e. Estas otras declaraciones son hipótesis de la

regla, si las hipótesis son correctas, entonces la declaración de abajo de la barra es verdadera. En conclusión, el *turnstile* ("$\vdash$") separa el contexto (O, M) de la declaración $(e : T)$.

Las reglas de inferencia más simples son las constantes

$$\frac{i \text{ es una literal entera}}{O, M \vdash i : int} \quad [\text{INTNUMBER}]$$

$$\frac{s \text{ es una literal string}}{O, M \vdash s : str} \quad [\text{STRING}]$$

$$\frac{}{O, M \vdash true : bool} \quad [\text{TRUE}]$$

$$\frac{}{O, M \vdash false : bool} \quad [\text{FALSE}]$$

La regla para los identificadores es simple también, si el entorno O le asigna al identificador id el tipo T, entonces el id tiene tipo T

$$\frac{O(id) = T}{O, M \vdash id : T} \quad [\text{ID}]$$

Las reglas para las comparaciones, aritmética y operadores lógicos son simples

$$\frac{\begin{array}{c} O, M \vdash e_1 : int \\ O, M \vdash e_2 : int \\ op \in \{+, -, *, /, \%\} \end{array}}{O, M \vdash e_1 \;\; op \;\; e_2 : int} \quad [\text{ARITHINT}]$$

$$\frac{\begin{array}{c} O, M \vdash e_1 : int \\ O, M \vdash e_2 : int \\ op \in \{<, <=\} \end{array}}{O, M \vdash e_1 \;\; op \;\; e_2 : bool} \quad [\text{COMPINT}]$$

$$\frac{O, M \vdash e_1 : bool}{O, M \vdash not \;\; e_1 : bool} \quad [\text{NOT}]$$

La regla de igualdad solo permite comparaciones entre los mismos tipos:

$$\frac{\begin{array}{c} O, M \vdash e_1 : T_1 \\ O, M \vdash e_2 : T_2 \\ T_1 = T_2 \end{array}}{O, M \vdash e_1 = e_2 : bool} \quad [\text{EQUAL}]$$

A pesar que el `if` y el `while` no son expresiones, es importante verificar el predicado del `if` y la condición del `while` sean de tipo `bool`.

$$\frac{O, M \vdash e_1 : bool}{O, M \vdash while \; (\; e_1 \;) \; \{ < statements > \}} \quad [\text{WHILE}]$$

$$\frac{O, M \vdash e_1 : bool}{O, M \vdash if \; (\; e_1 \;) \; \{ < statements > \} \; else \; \{ < statements > \}} \quad [\text{IF}]$$

En la regla para la declaración de identificadores locales, hay que tomar en cuenta que todo se revisa en un entorno modificado $O' = O[T/x]$ después de esa declaración.

$$\frac{\begin{array}{c} O(x) = T_0 \\ O, M \vdash e_1 : T_1 \\ T_1 = T_0 \end{array}}{O', M \vdash T_0 \;\; x = e_1} \qquad \text{[LocalDeclaration]}$$

La regla para la asignación es un poco compleja porque involucra ver en el entorno de tipos de identificadores

$$\frac{\begin{array}{c} O(id) = T \\ O, M \vdash e_1 : T' \\ T' = T \end{array}}{O, M \vdash id = e_1 : T'} \qquad \text{[Assign]}$$

El `return` es simple, ya que el tipo de la expresión que se quiere retornar es igual al del `return`

$$\frac{O, M \vdash e_1 : T}{O, M \vdash return\ e_1 : T} \qquad \text{[Return]}$$

La llamada a una función tal vez es la más difícil de verificar, pero solo por la parte de verificar que cada argumento sea del tipo definido en los parámetros.

$$\frac{\begin{array}{c} O, M \vdash e_1 : T_1 \\ O, M \vdash e_2 : T_2 \\ \vdots \\ O, M \vdash e_n : T_n \\ M(f) = (T'_1, ..., T'_n, T'_{n+1}) \\ T_i = T'_i \quad 1 \leq i \leq n \end{array}}{O, M \vdash f(e_1, ..., e_n) : T'_{n+1}} \qquad \text{[Call]}$$

Las dos reglas para las funciones, una sin `return` y la otra con

$$\frac{\begin{array}{c} M(f) = (T_1, ..., T_n, T_0) \\ T_0 = void \end{array}}{O, M \vdash def\ f(x_1 : T_1, ..., x_n : T_n) : T_0\ \{ < statements > \}} \qquad \text{[FunctionVoid]}$$

$$\frac{\begin{array}{c} M(f) = (T_1, ..., T_n, T_0) \\ O[T_1/x_1]...[T_n/x_n], M \vdash return\ e : T'_0 \\ T_0 \neq void \\ T_0 = T'_0 \end{array}}{O, M \vdash def\ f(x_1 : T_1, ..., x_n : T_n) : T_0\ \{ < statements >\ return\ e\}} \qquad \text{[Function]}$$

Donde aparece $< statements >$ son las declaraciones que se pueden definir por la gramática del lenguaje Viper, como sabemos que cada declaración tiene su propia regla de inferencia (como el `if`, el `while` o incluso alguna expresión como la suma) podemos decir que si hay n declaraciones, para cada declaración S_i con $1 <= i <= n$, aplicamos su regla de inferencia correspondiente.