Viper Language Type Rules

CC4 - Compiladores
Universidad Galileo

1 Type Rules

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

1.1 Type Environments

La razon de tener type environments es debido al problema que se encuentra en el caso de una expresion v, donde v es un identificador. No es posible decir que tipo tiene v, necesitamos saber el tipo con el que fue declarado v. Esa declaracion tiene que existir para cada identificador local en un programa valido de Viper.

Para capturar la informacion acerca de los tipos de los identificadores, nosotros usamos un *type environment*. El environment consiste de 2 partes: un environment de funciones M y un environment de identificadores O. El environment de funciones y el de identificadores son funciones (tambien llamados mappings). El environment de identificadores tiene la forma:

$$O(v) = T$$

que asigna el tipo T al identificador v. El environment de funciones es mas complejo, es una funcion de la forma:

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

donde f es el nombre de la funcion y $T_1, ..., T_n$ son tipos. La tupla de tipos es la firma de la funcion. La interpretacion de las firmas es que la funcion f tiene parametros definidos de tipo $(T_1, ..., T_{n-1})$ (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 funcion y un identificador con el mismo nombre.

A cada expresion e se le verifica su tipo en un type environment, cada subexpresion de e se le verifica su tipo en el mismo type environment o, si se introduce un nuevo identificador, en un environment modificado.
Consideren lo siguiente:

```
int c = 33;
...
```
la declaracion local de un identificador (que es un statement, no una expresion), introduce una nueva variable `c` de tipo `int`. Si O es el environment de identificadores donde aparecio el statement, entonces lo que sigue despues es verificado en el type environment:

$$O[int/c]$$

donde la notacion $O[T/c]$ 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 Type Checking Rules

La forma general de una regla de inferencia es:

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

La regla de inferencia se deberia de leer: En el environment de identificadores O y funciones M, la expresion e tiene tipo T. Los puntos arriba de la barra horizontal es donde van los statements acerca de las subexpresiones de e. Estos otros statements son hipotesis de la regla, si las hipotesis son correctas, entonces el statement abajo de la barra es verdadero. En conclusion, el "turnstile" ("$\vdash$") separa el contexto (O, M) del statement $(e : T)$.

Las reglas de inferencia mas simples son las constantes, aqui no hay pierde

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

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

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

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

La regla para los identificadores es simple tambien, si el environment 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, aritmetica y operadores logicos son simples

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

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

$$\frac{\begin{array}{c} O, M \vdash e_1 : bool \\ O, M \vdash e_2 : bool \\ op \in \{and, or\} \end{array}}{O, M \vdash e_1 \;\; op \;\; e_2 : bool} \quad [\text{LOGICAL}]$$

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

Para el while y el if lo unico que nos importa es que el *predicate/condition* sea de tipo bool para que sea correcto

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

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

La regla del print casi no tiene sentido, como es un statement solo tenemos que saber de que tipo es la expresion del argumento.

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

Las 2 reglas para la declaracion de identificadores locales, la primera cuando no se inicializa y la segunda cuando si. Hay que tomar en cuenta que a partir de aqui todo se revisa en un $O' = O[T/x]$

$$\frac{O(x) = T}{O', M \vdash T\ \ x} \qquad \text{[LocalDeclaration no init]}$$

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

La regla para la asignacion es un poco compleja porque involucra ver en el type environment de identificadores

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

El return es tan simple como el print, ya que el tipo de la expresion que se quiere retornar es igual al del return. Esto por el hecho de que el return es una expresion en este lenguaje en vez de un statement.

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

La llamada a una funcion tal vez es la mas dificil de verificar, pero solo

por la parte de verificar que cada Actual sea del tipo definido en los Formals.

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

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

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

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

Donde aparece un $< statements >$ son los statements que se pueden definir por la gramatica del lenguaje, como sabemos que cada statemet tiene su regla de inferencia podemos decir que si hay n statements, para cada statement S_i con $1 <= i <= n$, aplicamos su regla de inferencia correspondiente.