PyTiger2C
Anotaciones sobre la comprobación semántica

Yasser González Fernández
yglez@uh.cu

Ariel Hernández Amador
gnuaha7@uh.cu

1. Expresiones

1.1. Operadores

En el lenguaje Tiger los operadores binarios permitidos son `+ - * / = <> < <= > => & |`, agrupándolos de la manera convencional mediante el uso de paréntesis.

Los operadores binarios aritméticos son `+ - *` y `/` sólo pueden ser aplicados entre enteros y retornan el valor entero resultante.

Los operadores binarios de comparación `< <= >` y `=>` comparan sus operandos y pueden ser aplicados entre cadenas de caracteres o enteros, comparando las cadenas de caracteres del modo lexicográfico usual. Estos operadores retornan 1 si la comparación es verdadera y 0 en otro caso.

Los operadores binarios de igualdad `=` y `<>` comparan operandos del mismo tipo retornando 1 si la comparación es verdadera y 0 en otro caso. En el caso de la comparación entre cadena de caracteres, se comparan sus valores. En caso de los *records* está permitido compararlos con `nil` o con otro *record* del mismo tipo, comparándolos por referencia y no por valor. En el caso de los *array* también se realiza una comparación por referencia y no por valor.

Los operadores lógicos `|` y `&` son evaluados en cortocircuito (el operando de la derecha no se evalúa si el operando de la izquierda determina el valor), y sólo pueden ser aplicados entre enteros.

Los operadores aritméticos son asociativos a la izquierda de modo convencional, sin embargo los operadores de comparación no son asociativos, por lo que `a = b = c` sería ilegal.

1.2. Estructuras de control

Las estructuras de control disponibles en el lenguaje Tiger son las condicionales *if-then* e *if-then-else*, las estructuras de ciclos *while-do* y *for-do* y la estructura *break*.

La estructura *if-then* sintácticamente expresada como `if expr1 then expr2` evalúa la expresión `expr1` que debe retornar un valor entero y si su resultado es distinto de cero entonces se evalúa la expresión `expr2` que no debe retornar valor. La estructura *if-then* no retorna valor.

La estructura *if-then-else* sintácticamente expresada como `if expr1 then expr2 else expr3` evalúa la expresión `expr1` que debe retornar un valor entero y si su resultado es distinto de cero entonces se evalúa la expresión `expr2`, en caso contrario se evalúa la expresión `expr3`. En caso de que una de las expresiones `expr2` o `expr3` retornen valor, entonces ambas deben retornar valor y el tipo de retorno de ambos debe ser del mismo tipo. La estructura *if-then-else* retorna valor si ambas expresiones retornan valor y su tipo de retorno es el mismo que ambas expresiones.

La estructura *while-do* sintácticamente expresada como `while expr1 do expr2` evalúa la expresión `expr1`, que debe retornar un valor entero y si su resultado es distinto de cero entonces se evalúa la expresión `expr2` que no debe retornar valor. Este proceso se realizará mientras la expresión `expr1` sea evaluada y su valor sea distinto de cero. La estructura *while-do* no retorna valor.

La estructura *for-do* sintácticamente expresada como `for ID := expr1 to expr2 do expr3` declara una variable de tipo entera nombrada con el valor del *token* `ID`, esta variable toma el valor de retorno de la expresión `expr1` y solo esta definida en la expresión `expr3`. Luego se evalúa la expresión `expr3` que no debe retornar valor y que además no puede modificar el valor de la variable anteriormente mencionada, una vez evaluada esta, la variable aumenta su valor en 1. Este proceso se repite hasta que la variable alcanza el valor retornado por la expresión `expr3`. La estructura *for-do* no retorna valor.

La estructura *break* termina el ciclo en el que esté contenida, por lo tanto debe esta contenida en un ciclo. Esta estructura no puede estar contenida en ninguna estructura que retorne valor y ella misma no retorna valor.

1.3. Let

La estructura *let-in-end* sintácticamente expresada como `let declaration_list in expr end` realiza la declaración de los tipos, variables y funciones contenidas en la lista de declaraciones `declaration_list`

que estarán definidas sólo en la evaluación de la expresión `expr`. Luego se evalúa la expresión `expr`. La estructura *let-in-end* retorna valor si la expresión `expr` retrona valor y el valor de retorno de ambas es el mismo.

2. Declaraciones

2.1. Variables

En el lenguaje Tiger las declaraciones de variables permiten al programador especificar de manera explícita el tipo de esta o dejar que el compilador infiera el tipo de la misma.

La declaración de manera explícita del tipo de la variable, sintácticamente expresada como `var ID1 : ID2 := expr` declara una variable con el nombre `ID1` que tendrá tipo `ID2` y que tomará como valor el valor de retorno de la expresión `expr`. La expresión `expr` debe tener `ID2` como tipo de retorno, en caso de que el tipo `ID2` sea un *record*, entonces la expresión `expr` puede ser `nil` también. Esta es la única manera de declarar una variable con `nil` como valor.

La declaración de variables con inferencia de tipos, sintácticamente expresada como `var ID1 := expr` declara una variable con el nombre `ID1` que tendrá el mismo tipo de retorno que la expresión `expr` y su valor sera el que retorne la misma. La expresión `expr` no puede ser `nil` pues en ese caso no se podría determinar su tipo de retorno.

2.2. Funciones

En el lenguaje Tiger las declaraciones de funciones permiten al programador declarar funciones que retornen valor o no, en el caso de que las funciones no retornen valor las llamaremos *procedimientos*.

La declaración de un procedimiento, sintácticamente expresada como `function ID (field_types) = expr` declara una función de nombre `ID`, que tiene como parámetros aquellos especificados en la lista `field_types` que son pares del tipo `ID1 : ID2` separados por coma donde `ID1` es el nombre del parámetro e `ID2` es el tipo del mismo. La expresión `expr` corresponde al cuerpo de la función y esta no debe retornar valor. Un procedimiento no retorna valor.

La declaración de una función, sintácticamente expresada como `function ID1 (field_types) : ID2 = expr` declara una función con nombre `ID1`, que tiene como parámetros aquellos especificados en la lista `field_types`. El tipo de retorno de esta función será `ID2` y debe ser el mismo que el tipo de retorno de la expresión `expr` que corresponderá al cuerpo de la función.

Una secuencia de declaraciones de funciones (sin declaraciones de variables o variables entre ellas) puede ser mutuamente recursiva. En otro caso las funciones no pueden ser mutuamente recursivas.

2.3. Tipos

Los tipos en Tiger pueden ser declarados de 3 modos distintos, mediante *alias*, *array* y *records*.

Los *alias*, definidos sintácticamente como `type ID1 = ID2` definen un tipo con nombre `ID1` que es equivalente al tipo con nombre `ID2`. Una instancia de `ID1` y una instancia de `ID2` son perfectamente intercambiables, pues su única diferencia es en el nombre de su tipo.

Los *array*, definidos sintácticamente como `type ID1 = array of ID2` definen un tipo con nombre `ID1` que será una lista de tamaño fijo de elementos del tipo `ID2`. Dos instancias de tipos *array* distintos siempre serán distintas.

Los *records*, definidos sintácticamente como `type ID = {field_types}` definen un tipo con nombre `ID` que tendrá los campos contenidos en la lista `field_types`, que sigue el mismo formato que los parámetros de las funciones. Dos instancias de *records* distintos siempre serán distintas.

Una secuencia de declaraciones de tipos (sin declaraciones de funciones o variables entre ellas) puede ser mutuamente recursiva. En otro caso los tipos no pueden ser mutuamente recursivos.