Lenguajes de Programación

Práctica 6

y puntos de recuperación

Karla Ramírez Pulido Manuel Soto Romero Alejandro Hernández Mora
Alma Rocío Sánchez Salgado Silvia Díaz Gómez

Semestre 2021-1

Facultad de Ciencias, UNAM

Fecha de inicio: 27 de enero de 2021

Fecha de entrega: 9 de febrero de 2021

1. Objetivos

Implementar el verificador de tipos para el lenguaje ***Typed-CFWBAEL***. Para llevar a cabo esta tarea, se debe completar el cuerpo de las funciones faltantes dentro de los archivos ***grammars.rkt***, ***parser.rkt*** y ***verificador.rkt***[1].

La gramática del lenguaje `Typed-CFBWAE` se presenta a continuación:

```
<expr> ::= <id>
         | <num>
         | <bool>
         | {<op> <expr>+}
         | {if <expr> <expr> <expr>}
         | {cond {<expr> <expr>}+ {else <expr>}}
         | {with {{<id> : <type>  <expr>}+} <expr>}
         | {with* {{<id> : <type> <expr>}+} <expr>}
         | {fun {{<id> : <type>}+}: <type> <expr>}
         | {<expr>}{<expr>*}
<id>    ::= a | b | c | ...
<num>   ::= 1 | 2 | 3 | ...
<bool>  ::= true | false
<op>    ::= + | - | * | / | modulo | expt | add1 | sub1
         | < | <= | = | > | >= | not | and | or | zero?
<type> ::= number | boolean | (<type>+ -> <type>)
```

[1] Puedes reutilizar los archivos que entregaste en la práctica anterior, agregando las modificaciones incluidas en esta práctica

La gramática anterior se define mediante los siguientes tipos en *Racket*.

```
;;Data-type que define al tipo de dato Type
(define-type Type
[numberT]
[booleanT]
[funT (params (listof Type?))])

;; Definición del tipo Type-Context
(define-type Type-Context
[phi]
[gamma (id symbol?) (tipo Type?) (rest Type-Context?)])

;; Definición del tipo Binding
(define-type Binding
[binding (id symbol?) (tipo Type?) (value SCFBWAE?)])

;; Definición del tipo Param
(define-type Param
[param (param symbol?) (tipo Type?)])

;;Definición del tipo condition para la definición de cond.
(define-type Condition
[condition (test-expr SCFBWAE?) (then-expr SCFBWAE?)]
[else-cond (else-expr SCFBWAE?)])

;; Definición del tipo SCFBWAE
(define-type SCFBWAE
[idS    (i symbol?)]
[numS   (n number?)]
[boolS  (b boolean?)]
[iFS    (condicion SCFBWAE?) (then SCFBWAE?) (else SCFBWAE?)]
[opS    (f procedure?) (args (listof SCFBWAE?))]
[condS  (cases (listof Condition?))]
[withS  (bindings (listof binding?)) (body SCFBWAE?)]
[withS* (bindings (listof binding?)) (body SCFBWAE?)]
[funS   (params (listof param?)) (rType Type?) (body SCFBWAE?)]
[appS   (fun SCFBWAE?) (args (listof SCFBWAE?))])
```

2. Ejercicios

1. (2 pts.) Modificar el cuerpo de la función `(parse sexp)` dentro del archivo `parser.rkt`, de manera que haga el análisis sintáctico.

```
;; parse: s-expression -> SCFBWAE
(define (parse sexp) ...)
```

   La función `parse` debe tomar un símbolo o una lista de elementos y transformarlo a una expresión del lenguaje ***SCFBWAE*** [2], que será representado mediante la sintaxis abstracta, definida anteriormente en el archivo ***grammars.rkt***. Esta sintaxis abstracta considera explícitamente los tipos de algunas expresiones, como se muestra en la gramática del lenguaje.

2. (8 pts.) Completar el cuerpo de la función `(typeof sexpr context)`, que se encuentra dentro del archivo `verificador.rkt`.

```
;; typeof SCFBWAE-> Type-Context -> Type
(define (typeof sexpr context)...)
```

   Dicha función debe tomar una expresión del tipo ***SCFBWAE***, verificar que no tenga errores de tipos y devolver el tipo del valor de retorno de cada expresión. En caso de que el programa no cumpla con alguna de las dos condiciones anteriores se debe mandar un error de tipos.

   - **Operaciones aritméticas y lógicas (op):**
     Se debe verificar que el valor de cada parámetro recibido en la lista tenga el tipo correspondiente según el operador. Para los operadores `+`, `-`, `*`, `/`, `=`, `modulo`, `expt`, `add1`, `sub1`, `<`, `<=`, `>`, `>=`, `>`, `zero?` los parámetros deben tener tipo `numberT`. Para los operadores `and`, `or` y `not` los parámetros deben tener tipo `booleanT`.
   - **Condicionales (if, cond):**
     Se debe verificar que las condicionales en ambos casos sea algo de tipo `booleanT` o mandar un error de tipos en otro caso. Además en ambos casos los tipos de las expresiones a evaluar deben ser el mismo para todas las posibles expresiones que se van a ejecutar; es decir que en un `if` la condicional tiene un tipo `booleanT` y las expresiones `then-expression` y `else-expression` deben tener el mismo tipo. Análogamente esto debe suceder con las expresiones de tipo `cond`, conde las `test-expression` tengan tipo `booleanT` y tanto las `then-expression` como las `else-expression` tengan el mismo tipo. Si las expresiones no tienen el mismo tipo se debe mandar un error de tipos[3].
   - **Funciones (fun):**
     Se debe guardar dentro del contexto el tipo de cada uno de los parámetros recibidos en la función, para que al aplicar la función, se puedan obtener los tipos de cada uno de éstos. La función tendrá tipo `funT` $(a_1, a_2, ..., r)$, donde cada a_i representa el tipo del parámetro recibido en la posición i, y r es el tipo del valor de retorno de la función.

[2] La ***S*** es de *sugar*, se explicará por qué en esta práctica aunque no es necesario quitar el azúcar sintáctica de la sintaxis abstracta se decidió dejar la ***S***, pon atención a este detalle, porque podrías ganar puntos extra.

[3] Esto es una decisión de diseño de lenguaje, que tiene sus diferentes implicaciones, tanto ventajas como desventajas. Por esta razón, no todos los lenguajes funcionan de esta manera.

- **Asignaciones locales simples y anidadas (*with, with**):**
  Se debe verificar que los parámetros recibidos se evalúen al mismo tipo al que fueron declarados.
- **Aplicaciones de función (app):**
  Verificar que la función tenga el tipo `funT` y que los parámetros de la función tengan el tipo correspondiente a la declaración de la función.

3. Puntos extra

Los puntos extra serán retroactivos a alguna práctica donde necesiten el puntaje para mejorar su calificación. Los puntos extra se sumarán a una sola práctica y el puntaje máximo será de 10.

1. (2 pts.) Integra el verificador de tipos a la práctica anterior, de manera que en un mismo programa se ejecute la verificación de tipos y se realice la interpretación de las expresiones, cumpliendo las especificaciones de ésta y la práctica anterior.
2. (1 pts.) Encuentra dos lenguajes que hagan diferente manejo de tipos entre sí en las condicionales *if* y *cond* (o similares), donde expliques si el lenguaje permite diferentes tipos en las múltiples expresiones que se pueden evaluar, dependiendo de si se cumple o no la condicional. Adicionalmente muestra tres instrucciones que ejemplifiquen la explicación anterior. Deberás adjuntar un archivo en formato *PDF* con este ejercicio.

4. Requerimientos

Se deben subir los archivos requeridos a la plataforma google classroom antes de las 23:59 hrs del día de la fecha de entrega, conforme lo como lo indican los lineamientos de entrega. No olvides incluir el archivo `ReadMe.txt` con los datos de tus compañeros. Sólo es necesario que una persona suba la práctica, los demás compañeros deberán subir como tarea el contenido del archivo `ReadMe.txt`, de manera que se pueda ver el equipo que integran desde la plataforma google classroom.

El orden en el que aparezcan las funciones en los archivos solicitados, debe ser el orden especificado en este archivo PDF, de lo contrario podrán penalizarse algunos puntos.

La idea es que reutilices funciones de los ejercicios que ya se programaron anteriormente.

¡Que tengas éxito en tu práctica!.