Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI4721 - Lenguajes de Programación II
Trimestre Abril - Julio 2021
Prof. Ricardo Monascal
Amin Arriaga 16-10072

Examen I

(20 puntos)

El repositorio de Github usado para este examen se encuentrá en este link.

1. *(6 puntos)* Sea una gramática `EXCEP T = ({instr, ;, try, catch, f inally}, {I}, P, I)`, con `P` definido de la siguiente manera:

```
I  -> try I catch I finally I
    | try I catch I
    | I ; I
    | instr
```

   Esto representa instrucciones con bloques protegidos (`try`) y manejadores de excepciones (`catch`), que opcionalmente tienen una instrucción que se ejecuta en cualquier caso, ya sea que ocurriera la excepción o no (`finally`).

   (a) *(3 puntos)* Aumente la gramática con un nuevo símbolo inicial no recursivo `S`, construya la máquina característica *LR(1)* y diga si existen conflictos en el mismo
   Respuesta: Se colocó la nueva gramática, la máquina característica y sus conflictos en el siguiente archivo del repositorio.

   (b) *(1 punto)* Tome en consideración las siguientes reglas:
   - `;` asocia a la izquierda.
   - `finally` se asocia al `try` más interno.
   - `finally` tiene más precedencia que `;`.
   - `catch` tiene más precedencia que `;`

   En caso de haber conflictos en el autómata de prefijos viables *LR(1)*, diga cómo resolverá los conflictos (seleccionando una de las acciones que conforma dicho conflicto), de tal forma que las reglas anteriores sean satisfechas.
   Respuesta: Los conflictos se resolvieron en el mismo archivo de la pregunta anterior.

   (c) *(2 puntos)* A partir de las respuestas anteriores, construya la máquina caracteríistica *LALR(1)* y diga si existen conflictos en el mismo. En caso de existir, explique cómo los resolvería (seleccionando una de las acciones que conforma dicho conflicto), con las mismas reglas de la pregunta anterior.
   Respuesta: Se colocó la máquina característica, sus conflictos y la forma de resolverlos en el siguiente archivo del repositorio.

2. *(5 puntos)* Considerando la misma gramática de la pregunta anterior:

   (a) *(2 puntos)* Proponga una Relación de Precedencia de Operadores entre los símbolos terminales de la gramática que permita resolver los conflictos con la misma suposición que en las preguntas anteriores.
   Respuesta:

	try	catch	finally	;	instr	$
try	<	=			<	
catch	<	>	=	<	<	>
finally	<	>	>	>	<	>
;	<			>	<	>
instr		>	>	>		>
$	<			<	<	

   (b) *(1.5 puntos)* Use el reconocedor para reconocer la frase:

   `instr ; try instr catch instr ; try instr catch instr f inally instr ; instr`

   Respuesta: Primero enumeramos las producciones

```
(0)    S  -> I $
(1)    I  -> try I catch I finally I
(2)        | try I catch I
(3)        | I ; I
(4)        | instr
```

ENTRADA	**PILA**	**ACCIÓN**
instr ; try instr catch instr ; try instr catch instr finally instr ; instr $	$ I0	SHIFT
; try instr catch instr ; try instr catch instr finally instr ; instr $	$ I0 I1	REDUCE (4)
; try instr catch instr ; try instr catch instr finally instr ; instr $	$ I0 I2	SHIFT
try instr catch instr ; try instr catch instr finally instr ; instr $	$ I0 I2 I4	SHIFT
instr catch instr ; try instr catch instr finally instr ; instr $	$ I0 I2 I4 I3	SHIFT
catch instr ; try instr catch instr finally instr ; instr $	$ I0 I2 I4 I3 I1	REDUCE (4)
catch instr ; try instr catch instr finally instr ; instr $	$ I0 I2 I4 I3 I5	SHIFT
instr ; try instr catch instr finally instr ; instr $	$ I0 I2 I4 I3 I5 I7	SHIFT
; try instr catch instr finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I1	REDUCE (4)
; try instr catch instr finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8	SHIFT

try instr catch instr finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4	SHIFT
instr catch instr finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3	SHIFT
catch instr finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3 I1	REDUCE (4)
catch instr finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3 I5	SHIFT
instr finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3 I5 I7	SHIFT
finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3 I5 I7 I1	REDUCE (4)
finally instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3 I5 I7 I8	SHIFT
instr ; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3 I5 I7 I8 I9	SHIFT
; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3 I5 I7 I8 I9 I1	REDUCE (4)
; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I3 I5 I7 I8 I9 I10	REDUCE (1)
; instr $	$ I0 I2 I4 I3 I5 I7 I8 I4 I6	REDUCE (3)
; instr $	$ I0 I2 I4 I3 I5 I7 I8	SHIFT
instr $	$ I0 I2 I4 I3 I5 I7 I8 I4	SHIFT
$	$ I0 I2 I4 I3 I5 I7 I8 I4 I1	REDUCE (4)
$	$ I0 I2 I4 I3 I5 I7 I8	REDUCE (2)
$	$ I0 I2 I4 I6	REDUCE (3)
$	$ I0 I2	SHIFT
	$ I0 I2 I11	ACCEPT

(c) *(1.5 puntos)* Calcule las funciones de precedencia `f` y `g` según el algoritmo estudiado en clase (o argumente por qué dichas funciones no pueden ser construidas).

Respuesta: Representaremos el grafo colocando cada nodo (clase de equivalencia) apuntando al conjunto de sus nodos sucesores:

NODO		SUCESORES
$\{G_{instr}\}$	$\Longrightarrow$	$\{\{F_{try}, G_{catch}\}, \{F_{\$}\}, \{F_{catch}, G_{finally}\}, \{F_{finally}\}, \{F_{;}\}\}$
$\{G_{try}\}$	$\Longrightarrow$	$\{\{F_{try}, G_{catch}\}, \{F_{\$}\}, \{F_{catch}, G_{finally}\}, \{F_{finally}\}, \{F_{;}\}\}$
$\{F_{instr}\}$	$\Longrightarrow$	$\{\{F_{try}, G_{catch}\}, \{G_{\$}\}, \{F_{catch}, G_{finally}\}, \{G_{;}\}\}$
$\{F_{;}\}$	$\Longrightarrow$	$\{\{G_{\$}\}, \{G_{;}\}\}$
$\{F_{finally}\}$	$\Longrightarrow$	$\{\{F_{try}, G_{catch}\}, \{G_{\$}\}, \{F_{catch}, G_{finally}\}, \{G_{;}\}\}$
$\{G_{;}\}$	$\Longrightarrow$	$\{\{F_{catch}, G_{finally}\}, \{F_{\$}\}\}$
$\{F_{catch}, G_{finally}\}$	$\Longrightarrow$	$\{\{F_{try}, G_{catch}\}, \{G_{\$}\}\}$
$\{F_{\$}\}$	$\Longrightarrow$	$\emptyset$
$\{G_{\$}\}$	$\Longrightarrow$	$\emptyset$
$\{F_{try}, G_{catch}\}$	$\Longrightarrow$	$\emptyset$

Así, los valores de `f` y `g` son:

	try	catch	finally	;	instr	$
f	0	1	3	3	3	0
g	4	0	1	2	4	0

3. *(4 puntos)* Considererando la misma gramática de las preguntas anteriores, implementaremos una semántica para este lenguaje donde las instrucciones tienen un valor además del efecto de borde que ocasionan. Para cualquier instrucción, su valor será el valor de la última expresión que haya sido evaluada (o de la última instrucción ejecutada, equivalentemente).

   (a) *(2 puntos)* Aumente el símbolo no-terminal `I` con un atributo `tipo`, que contenga el tipo del valor que retorna la instrucción representada en `I`. Puede suponer que cuenta con un tipo `Either A B` para representar un tipo que es opcionalmente `A` o `B`. Puede suponer, además, que el sıímbolo `instr` tiene un atributo intrínseco tipo que tiene el tipo para ese terminal. Puede agregar todos los atributos adicionales que desee a `I`, cuidando que la gramática resultante sea *S-atribuida*.

   Respuesta:

```
S   ->  I $                          { S.tipo <- I.tipo }
I   ->  try I0 catch I1 finally I2   { I.tipo <- I2.tipo }
     |  try I0 catch I1              { I.tipo <- Either I0.tipo I1.tipo }
     |  I0 ; I1                      { I.tipo <- I1.tipo }
     |  instr                        { I.tipo <- instr.tipo }
```

(b) *(1 punto)* Tenemos a nuestra disposición un reconocedor descendente. La gramática anterior tiene prefijos comunes y recursión izquierda. Transforme la gramática de tal forma que sea apropiada para un reconocedor descendente. Recuerde agregar atributos y reglas de tal forma que aún se calcule el tipo de la instrucción en tipo, cuidando que la gramática resultante sea L–atribuida.

Respuesta:

```
S   ->  I { S.tipo <- I.tipo } $

I   ->  E { R.in <- E.tipo } R { I.tipo <- R.tipo }

R   ->  ; I { R0.in <- I.tipo } R0 { R.tipo <- R0.tipo }
     |  { R.tipo <- R.in }

E   ->  try I0 catch { I1.in <- I0.tipo }
            I1 { F.in <- Either I1.in I1.tipo }
            F { E.tipo <- F.tipo }
     |  instr { E.tipo <- instr.tipo }

F   ->  finally E { F.tipo <- E.tipo }
     |  { F.tipo <- F.in }
```

(c) *(1 punto)* Construya un reconocedor recursivo descendente a partir de su gramática. Esto es, escriba las funciones (en el lenguaje de su elección) que reconozca frases en el lenguaje y calculen el atributo tipo para una instrucción bien formada. Deben llevar una variable *lookahead* que contenga el siguiente símbolo de la entrada en todo momento. Su programa debe funcionar correctamente para cualquier entrada y estar alojada en un repositorio *git* público.

Respuesta: La implementación del reconocedor recursivo descendente se encuentra en el siguiente archivo del repositorio. Debe correr el script con `python3`, el cual le proporcionará un prompt para usar el reconocedor, cuya sintaxis es:

```
$> PARSE [<string> ...]
$> SALIR
```

4. *(5 puntos)* Se desea que modele e implemente, en el lenguaje de su elección, un generador de analizadores sintácticos para gramáticas de operadores. Investigue herramientas para pruebas unitarias y cobertura en su lenguaje escogido y agregue pruebas a su programa que permitan corroborar su correcto funcionamiento. Como regla general, su programa debería tener una cobertura (de líneas de código y de bifuración) mayor al 80%.

Respuesta: La implementación del generador de analizadores sintácticos se encuentra en el siguiente archivo del repositorio.