Tarea 5:
Lenguajes de Programación

Araujo Chávez Mauricio
312210047

Carmona Mendoza Martín
313075977

Considera la gramática descrita en la tarea pasada.

1.
- Extiende la gramatica de la tarea pasada con los marcos de operación de las expresiones error y try e1 catch e2 vistas en clase y describe los marcos de operacion (no es necesario describir los marcos descritos en la tarea pasada).
error
Denotamos error como un nuevo estado de la máquina denotado por

$$\mathcal{P} \prec\!\!\prec error$$

Donde el erorr va vaciando la pila hasta encontrar un manejador y dado el caso que no existe la expresión anterior se considera como estado final.

$$Donde\ \mathcal{P}\ es\ la\ pila.$$

try e1 catch e2
Se ejecuta e1 y se espera un valor para manejar el error si es que existe.

$$\overline{catch(-, e_2)\ marco}$$

- Escribe cuatro programas utilizando la gramatica descrita. Todos deben estar contenidos en un try catch, dos de ellos deben arrojar un error (y manejarlo) y dos de ellos deben terminar sin utilizar el marco del catch. Entre los cuatro programas debes utilizar todos las expresiones del lenguaje.
  - $p_1 \leftrightharpoons$

```
try
        let x = 2 in
                x + (if not(x < 3) then 10 else error)
        end
catch error
        show "El numero es menor que 3"
```

  - $p_2 \leftrightharpoons$

```
try
        app(if((λx.(not x)true)
                then λy.(y+6)
                else λz(z+0)
                ,7)
catch error
        show "Valor incorrecto"
```

- Realiza la ejecucion de los cuatro programas, puedes obviar algunas operaciones excepto aquellas que involucran la expresion try-catch.
  - $p_1 \leftrightharpoons$
    Definimos e1 como let x = 2 in x + (if not(x < 3) then 10 else error) end
    y e2 como show "El numero es menor que 3"

$$\square \succ p_1 \rightarrow_{\mathcal{K}}$$

$$catch(-, e_2); \square \succ e_1 \rightarrow_{\mathcal{K}}$$

    Definimos l como x + (if not(x < 3) then 10 else 30)

$$(-, x.l), catch(-, e_2); \square \succ 2 \rightarrow_{\mathcal{K}}$$

$$(-, x.l), catch(-, e_2); \square \prec 2 \rightarrow_{\mathcal{K}}$$

    Obtenemos 2 + (if not(2 < 3) then 10 else error)

$$(2, -), catch(-, e_2); \square \prec (if\ not(2 < 3)\ then\ 10\ else\ error) \rightarrow_{\mathcal{K}}$$

$$if(-, 10, error); (2, -), catch(-, e_2); \square \prec not(2 < 3) \rightarrow_{\mathcal{K}}$$

Al tener valores omitimos pasos triviales de $<$

$$if(-, 10, error); (2, -), catch(-, e_2); \square \prec not\ true \rightarrow_{\mathcal{K}}$$

$$if(-, 10, error); (2, -), catch(-, e_2); \square \succ false \rightarrow_{\mathcal{K}}$$

$$if(-, 10, error); (2, -), catch(-, e_2); \square \prec false \rightarrow_{\mathcal{K}}$$

$$(2, -), catch(-, e_2); \square \succ error \rightarrow_{\mathcal{K}}$$

Propagamos el error

$$(2, -), catch(-, e_2); \square \prec\!\!\prec error \rightarrow_{\mathcal{K}}$$

Y al tener $catch(-, e_2)$ podemos continuar con la ejecución de e_2

$$catch(error, -)\square \succ e_2 \rightarrow_{\mathcal{K}}$$

Intuitivamente, el programa imprimirá la leyenda.

$$\square \succ \text{"}El\ numero\ es\ menor\ que\ 3\text{"}$$

– $p_2 \leftrightharpoons$
Definimos e1 como app(if(λx.(not x)true) then λy.(y+x) else λz(z+),7)
y e2 como show "Valor incorrecto"

$$\square \succ p_2 \rightarrow_{\mathcal{K}}$$

$$catch(-, e_2); \square \succ e_1 \rightarrow_{\mathcal{K}}$$

Definimos l como if(λx.(not x)true) then λy.(y+x) else λz(z+0)

$$app(-, 7), catch(-, e_2); \square \succ l \rightarrow_{\mathcal{K}}$$

$$if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \succ (\lambda x.(not\ x))true \rightarrow_{\mathcal{K}}$$

$$app(-, true); if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \succ \lambda x.(not\ x) \rightarrow_{\mathcal{K}}$$

$$app(-, true); if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \prec \lambda x.(not\ x) \rightarrow_{\mathcal{K}}$$

$$app(\lambda x.(not\ x), -); if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \succ true \rightarrow_{\mathcal{K}}$$

$$app(\lambda x.(not\ x), -); if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \prec true \rightarrow_{\mathcal{K}}$$

$$if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \succ (not\ x)[x\ :=\ true] \rightarrow_{\mathcal{K}}$$

$$if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \succ not\ true \rightarrow_{\mathcal{K}}$$

$$not(-); if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \succ true \rightarrow_{\mathcal{K}}$$

$$not(-); if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \prec true \rightarrow_{\mathcal{K}}$$

$$if(-, \lambda y.(y + x), \lambda z(z + 0); app(-, 7), catch(-, e_2); \square \prec false \rightarrow_{\mathcal{K}}$$

$$app(-, 7), catch(-, e_2); \square \succ \lambda z(z + 0) \rightarrow_{\mathcal{K}}$$

$$app(-, 7), catch(-, e_2); \square \prec \lambda z(z + 0) \rightarrow_{\mathcal{K}}$$

$$app(\lambda z(z + 0),), catch(-, e_2); \square \succ 7 \rightarrow_{\mathcal{K}}$$

$$app(\lambda z(z + 0),), catch(-, e_2); \square \prec 7 \rightarrow_{\mathcal{K}}$$

$$catch(-, e_2); \square \succ (z\ +\ 0)[z\ :=\ 7]$$

Trivialmente sumamos 7 + 0

$$catch(-, e_2); \square \succ 7 \rightarrow_{\mathcal{K}}$$

Al devolver un valor e1 se elimina el marco manejador y sigue ejecución sin salto.

$$\square \succ 7$$

2. • Extiende la gramatica de la tarea pasada con los marcos de operación de las expresiones $raise(e)$ y $handle\ e1\ with\ x \Rightarrow e2$ vistas en clase y describe los marcos de operacion (no es necesario describir los marcos descritos en la tarea pasada).
raise(e)
Lanzamos el error esperando un valor de e.

$$\frac{}{raise(-)\ marco}$$

handle e1 with x ⇒ e2
Manejamos un error con x aplicandolo a e2.

$$\frac{}{handle(-, x.e_2)\ marco}$$

• Extiende el comportamiento de la maquina $\mathcal{K}$
Extendemos de la siguiente manera.
raise

$$\frac{}{\mathcal{P} \succ raise(e) \rightarrow_{\mathcal{K}} raise(-);\ \mathcal{P} \succ e}$$

$$\frac{}{raise(-);\ \mathcal{P} \prec v \rightarrow_{\mathcal{K}} \mathcal{P} \prec\!\prec raise(v)}$$

handle

$$\frac{}{\mathcal{P} \succ handle(e_1, x.e_2) \rightarrow_{\mathcal{K}} handle(-, x.e_2);\ \mathcal{P} \succ e_1}$$

$$\frac{}{handle(-, x.e_2);\ \mathcal{P} \prec v \rightarrow_{\mathcal{K}} \mathcal{P} \prec v}$$

$$\frac{m \neq handle(-, x.e_2)}{m; \mathcal{P} \prec\!\prec raise(v) \rightarrow_{\mathcal{K}} \mathcal{P} \prec\!\prec raise(v)}$$

$$\frac{}{handle(-, x.e_2); \mathcal{P} \prec\!\prec raise(v) \rightarrow_{\mathcal{K}} \mathcal{P} \succ e_2[x\ :=\ v]}$$

• Escribe cuatro programas utilizando la gramatica descrita. Todos deben estar contenidos en un handle, dos de ellos deben arrojar un error con valor (y manejarlo) y dos de ellos deben terminar sin utilizar el manejador. Entre los cuatro programas debes utilizar todos las expresiones del lenguaje.

• Realiza la ejecucion de los cuatro programas, puedes obviar algunas operaciones excepto aquellas que involucran la expresiones raise y handle.