Lenguajes de Programación 2020-1
Facultad de Ciencias UNAM
Ejericio Semanal 11

Sandra del Mar Soto Corderi
Edgar Quiroz Castañeda

7 de noviembre de 2019

1. Considera el siguiente programa

```
exception Excpt of int ;
fun twice (f , x ) = handle f ( f ( x ) ) with Excpt ( x ) = > x ;
fun pred ( x ) = if iszero x then raise Excpt ( x ) else x - 1;
fun smart ( x ) = handle 1 + pred ( x ) with Excpt ( x ) = > 1;
```

Evalúa las siguientes expresiones utilizando la semántica operacional de la máquina K extendida para manejar excepciones con valor. (pueden obviarse algunos pasos pero no los que involucran manejo de excepciones. En cada caso describe qué excepción fue lanzada y en dónde.)

- `twice(smart, 0)`

  Cuando se tenga una aplicación con valores bloqueados, se saltarán los pasos hasta el cálculo que los concierna, ignorando los pasos intermedios para verificar que son valores.

$\square \succ$ `app(twice, (smart, 0))`
$\to_k \square \succ$ `twice[f := twice, x := 0]`
$\to_\beta \square \succ$ `handle(app(smart, app(smart, 0)), x.x)`
$\to_k$ `handle(_, x.x)` $\succ$ `app(smart, app(smart, 0))`
$\to_k$ `app(smart, _),handle(_, x.x)` $\succ$ `app(smart, 0)`
$\to_k$ `app(smart, _),handle(_, x.x)` $\succ$ `smart[x := 0]`
$\to_\beta$ `app(smart, _),handle(_, x.x)` $\succ$ `handle(sum(1, app(pred,0)), x.1)`
$\to_k$ `handle(_, x.1),app(smart, _),handle(_, x.x)` $\succ$ `sum(1, app(pred,0))`
$\to_k$ `sum(1, _),handle(_, x.1),app(smart, _),handle(_, x.x)` $\succ$ `app(pred,0)`
$\to_k$ `sum(1, _),handle(_, x.1),app(smart, _),handle(_, x.x)` $\succ$ `pred[x:=0]`
$\to_\beta$ `sum(1, _),handle(_, x.1),app(smart, _),handle(_, x.x)` $\succ$ `if(iszero(0), raise(0), dif(0, 1))`
$\to_k$ `if(_, raise(0), dif(0, 1)),sum(1, _),handle(_, x.1),app(smart, _),handle(_, x.x)` $\succ$ `iszero(0)`
$\to_k$ `if(_, raise(0), dif(0, 1)),sum(1, _),handle(_, x.1),app(smart, _),handle(_, x.x)` $\prec$ `true`
$\to_k$ `sum(1, _),handle(_, x.1),app(smart, _),handle(_, x.x)` $\ll$ `raise(0)`

En este punto se lanza una excepción de tipo `Excpt` con un valor de 0.

$\to_k$ `handle(_, x.1)`, `app(smart, _)`, `handle(_, x.x)` $\ll$ `raise(0)`
$\to_k$ `app(smart, _)`, `handle(_, x.x)` $\succ$ `1[x:=0]`
$\to_\beta$ `app(smart, _)`, `handle(_, x.x)` $\succ$ 1
$\to_k$ `handle(_, x.x)` $\succ$ `smart[x:=1]`
$\to_\beta$ `handle(_, x.x)` $\succ$ `handle(sum(1, app(pred,1)), x.1)`
$\to_k$ `handle(_, x.1)`, `handle(_, x.x)` $\succ$ `sum(1, app(pred,1))`
$\to_k$ `sum(1, _)`, `handle(_, x.1)`, `handle(_, x.x)` $\succ$ `app(pred,1)`
$\to_k$ `sum(1, _)`, `handle(_, x.1)`, `handle(_, x.x)` $\succ$ `pred[x:=1]`
$\to_\beta$ `sum(1, _)`, `handle(_, x.1)`, `handle(_, x.x)` $\succ$ `if(iszero(1), raise(1), dif(1, 1))`
$\to_k$ `if(_, raise(1), dif(1, 1))`, `sum(1, _)`, `handle(_, x.1)`, `handle(_, x.x)` $\succ$ `iszero(1)`
$\to_k$ `if(_, raise(1), dif(1, 1))`, `sum(1, _)`, `handle(_, x.1)`, `handle(_, x.x)` $\prec$ `false`
$\to_k$ `sum(1, _)`, `handle(_, x.1)`, `handle(_, x.x)` $\succ$ `dif(1, 1)`
$\to_k$ `sum(1, _)`, `handle(_, x.1)`, `handle(_, x.x)` $\prec$ 0
$\to_k$ `handle(_, x.1)`, `handle(_, x.x)` $\succ$ `sum(1, 0)`
$\to_k$ `handle(_, x.1)`, `handle(_, x.x)` $\prec$ 1
$\to_k$ `handle(_, x.x)` $\prec$ 1
$\to_k \square \prec 1$

- `twice(pred, 1)`
  Cuando se tenga una aplicación con valores bloqueados, se saltarán los pasos hasta el cálculo que los concierna, ignorando los pasos intermedios para verificar que son valores.

$\square \succ$ `app(twice, (pred, 1))`
$\to_k \square \succ$ `twice[f := pred, n := 1]`
$\to_\beta \square \succ$ `handle(app(pred, app(pred, 1)), x.x)`
$\to_k$ `handle(_, x.x)` $\succ$ `app(pred, app(pred, 1))`
$\to_k$ `app(pred, _)`, `handle(_, x.x)` $\succ$ `app(pred, 1)`
$\to_k$ `app(pred, _)`, `handle(_, x.x)` $\succ$ `pred[x:=1]`
$\to_\beta$ `app(pred, _)`, `handle(_, x.x)` $\succ$ `if(iszero(1), raise(1), dif(1, 1))`
$\to_k$ `if(_, raise(1), dif(1, 1))`, `app(pred, _)`, `handle(_, x.x)` $\succ$ `iszero(1)`
$\to_k$ `if(_, raise(1), dif(1, 1))`, `app(pred, _)`, `handle(_, x.x)` $\prec$ `false`
$\to_k$ `if(_, raise(1), dif(1, 1))`, `app(pred, _)`, `handle(_, x.x)` $\succ$ `iszero(1)`
$\to_k$ `app(pred, _)`, `handle(_, x.x)` $\succ$ `dif(1, 1)`
$\to_k$ `app(pred, _)`, `handle(_, x.x)` $\prec$ 0
$\to_k$ `handle(_, x.x)` $\succ$ `app(pred, 0)`
$\to_k$ `handle(_, x.x)` $\succ$ `pred[x:=0]`
$\to_\beta$ `handle(_, x.x)` $\succ$ `if(iszero(0), raise(0), dif(0, 1))`
$\to_k$ `if(_, raise(0), dif(0, 1))`, `handle(_, x.x)` $\succ$ `iszero(0)`
$\to_k$ `if(_, raise(0), dif(0, 1))`, `handle(_, x.x)` $\prec$ `true`
$\to_k$ `handle(_, x.x)` $\ll$ `raise(0)`
$\to_k \square \succ$ `x[x:=0]`
$\to_\beta \square \succ 0$
$\to_k \square \prec 0$