Abordaje Funcional a EDSLs

ECI 2024

Se desea implementar un pequeño EDSL para representar proposiciones basadas en comparaciones entre números naturales, el cual tiene las siguientes construcciones:

- `val`: valor natural
- `eq`: igualdad entre naturales
- `lt`: relación de menor entre naturales
- `not`: negación
- `and`: conjunción
- `or`: disyunción

Sistema de Tipos

Asociado al EDSL se tiene un sistema de tipos que está formado por las reglas que se presentan a continuación. Como es habitual, el juicio `p : t` significa que la expresión `p` tiene tipo `t`. El lenguaje maneja dos tipos (`integer` y `boolean`).

$$\frac{}{\vdash \texttt{val}\ n : \texttt{integer}} \quad \text{(Val)}$$

$$\frac{\vdash p_1 : \texttt{integer} \quad \vdash p_2 : \texttt{integer}}{\vdash \texttt{eq}\ p_1\ p_2 : \texttt{boolean}} \quad \text{(Eq)}$$

$$\frac{\vdash p_1 : \texttt{integer} \quad \vdash p_2 : \texttt{integer}}{\vdash \texttt{lt}\ p_1\ p_2 : \texttt{boolean}} \quad \text{(Lt)}$$

$$\frac{\vdash p : \texttt{boolean}}{\vdash \texttt{not}\ p : \texttt{boolean}} \quad \text{(Not)}$$

$$\frac{\vdash p_1 : \texttt{boolean} \quad \vdash p_2 : \texttt{boolean}}{\vdash \texttt{and}\ p_1\ p_2 : \texttt{boolean}} \quad \text{(And)}$$

$$\frac{\vdash p_1 : \texttt{boolean} \quad \vdash p_2 : \texttt{boolean}}{\vdash \texttt{or}\ p_1\ p_2 : \texttt{boolean}} \quad \text{(Or)}$$

Ejemplos de expresiones bien tipadas:

- `var 4 : integer`
- `lt (var 4) (var 5) : boolean`
- `or (not (lt (var 4) (var 3))) (eq (var 4) (var 3)) : boolean`

Ejemplos de expresiones no tipadas:

- `or (var 4) (var 5)`
- `not (var 4)`
- `eq (lt (var 4) (var 3)) (var 5)`

Requisitos

Considerando que la interpretación estándar de las proposiciones es su valor de verdad o el propio natural `n` en el caso de expresiones de la forma `val n`, se pide:

1. Implementar el EDSL como un **shallow embedding** bien tipado en Haskell siguiendo el enfoque *tagless-final*. Interprete los tipos `integer` y `boolean` como tipos de Haskell.
2. Implementar el EDSL como un **deep embedding** bien tipado en Haskell utilizando GADTs. Definir la función `eval` que evalúa una expresión bien tipada de tipo `t` y retorna un valor de ese tipo.
3. Dada la siguiente gramática que describe una sintaxis concreta para el lenguaje:

```
prop ::= term "\/" prop | term
term ::= factor "/\" term | factor
factor ::= '~' prop
        | '(' prop ')'
        | '(' prop '=' prop ')'
        | '(' prop '<' prop ')'
        | N
```

   - (a) Definir una nueva interpretación para los puntos 1 y 2 que retorne un `String` que represente el programa en esa sintaxis.
   - (b) Definir un tipo `UProp`, de kind `*`, que represente el árbol de sintaxis abstracta no tipado del lenguaje.
   - (c) Escribir un *parser* del lenguaje utilizando los combinadores vistos en el curso (puede optar por usar los combinadores aplicativos o monádicos) y que retorne el valor de tipo `UProp` correspondiente.

4. (OPCIONAL) Considere la siguiente extensión al lenguaje, en la que se agrega:

   - `var`: variable proposicional

   Con la siguiente regla de tipado:

   - `var x : boolean`

   Ahora la interpretación depende de un ambiente de variables en el que se le asocian valores de verdad a las variables.

   - (a) Extender el EDSL definido en 1.
   - (b) Extender el EDSL definido en 2.
   - (c) Extender el tipo definido en 3.a y el parser definido en 3.b.
   - (d) Definir una función `typeProp :: Ty t -> UProp -> Prop t` donde `Prop t` es el GADT que definió para el *deep embedding* y `Ty t` es el siguiente tipo que representa tipos:

```
data Ty :: * -> * where
    TInt :: Ty Int
    TBool :: Ty Bool
```