Tarea 0 - Especificación de Expresiones de Conjuntos (Finitos) de Enteros

Teoría de la Computación
Universidad ORT Uruguay

Agosto 2023

En las siguientes líneas presentamos la especificación de las *Expresiones de conjuntos finitos de enteros*.

0. Sintaxis: En este lenguaje contaremos con la noción de variable y asignación (para permitirnos almacenar resultados en una memoria), con expresiones booleanas como la de pertenencia de un elemento a un conjunto o la inclusión de un conjunto en otro, y finalmente con expresiones que nos permitan construir conjuntos, formando el conjunto vacío y los conjuntos unitarios, más funciones conocidas, como la unión, intersección y diferencia de conjuntos.

$$
\begin{array}{lll}
e_1, e_2 & ::= & x \quad \text{(variable)} \\
& | & \varnothing \quad \text{(conjunto vacío)} \\
& | & \{z\} \quad \text{(conjunto unitario)} \\
& | & z \in e_1 \quad \text{(pertenencia)} \\
& | & e_1 \cup e_2 \quad \text{(unión)} \\
& | & e_1 \cap e_2 \quad \text{(intersección)} \\
& | & e_1 - e_2 \quad \text{(diferencia)} \\
& | & e_1 \subseteq e_2 \quad \text{(inclusión)} \\
& | & x := e_1 \quad \text{(asignación)}
\end{array}
$$

$$
\begin{array}{lll}
x & ::= & \mathit{String} \quad \text{(identificador de las variables)} \\
z & ::= & \mathbb{Z} \quad \text{(enteros)}
\end{array}
$$

1. Semántica: En este apartado definiremos algunas estructuras auxiliares que serán necesarias para evaluar una expresión formada por la sintaxis definida previamente. Para empezar, estructuras como los *valores*, que pueden ser *booleanos* o *conjuntos finitos de enteros*. Además necesitaremos la noción de *memoria*, que será una *tabla* donde se almacenen las variables con su correspondiente valor. Al momento de definir la memoria es prudente y hasta necesario definir las operaciones que permiten interactuar sobre esta memoria, que serán la búsqueda y la actualización a las cuales llamaremos *lookup* y *update* respectivamente.
Finalmente, y como es de esperarse, será necesario presentar las reglas de evaluación de las expresiones en presencia de una memoria. Es importante remarcar que las reglas que definamos son las que establecen cómo obtener valores a partir de una expresión evaluada en una memoria. Esto quiere decir que no se definirán reglas para aquellos casos en donde la evaluación debería fallar, por ejemplo, no habrán reglas para expresiones que la sintaxis sí permite formar, pero que no tengan sentido como: $3 \in (1 \in \varnothing)$. (En este caso $1 \in \varnothing$ debería evaluar a `False` sin problema, pero lo que no tiene sentido es lo que se evaluaría justo después.)En otras palabras, para las expresiones que se evalúan correctamente estarán las reglas definidas en este apartado, en tanto la ausencia de reglas para ciertas expresiones significa que la evaluación de las mismas deberá fallar.

Valores: v ::= `[Int]` (conjuntos)
| b (booleanos)

Memoria: m ::= $\overline{(x, v)}$ (tabla clave-valor)

Serán necesarias las siguientes operaciones sobre la memoria:

- *Lkup (búsqueda: $x \overset{m}{\mapsto} v$):* Dada una variable `x` y una memoria `m`, `lkup x m` devuelve el valor asociado a `x` en `m`. En caso de que la variable no esté inicializada, retornará error.
- *Upd (actualización: $m \prec\!+ (x, v)$):* Dada una memoria `m` y una pareja variable-valor `(x,v)`, `upd m (x,v)` retorna la memoria `m` actualizando a la variable `x` con el valor `v`, en caso de que la `x` no se encontrara definida previamente, la define.

Reglas de evaluación: El juicio $M, e \Downarrow M', v$, donde M y M' son memorias, e una expresión y v una valor, se leerá: "La expresión e con la memoria M, evalúa al valor v, dejando la memoria actualizada M'".

$$\texttt{var}\ \frac{x \overset{M}{\mapsto} v}{M, x \Downarrow M, v} \qquad \texttt{empty}\ \frac{}{M, \varnothing \Downarrow M, [\,]} \qquad \texttt{unit}\ \frac{}{M, \{z\} \Downarrow M, [z]}$$

$$\texttt{pert}\ \frac{M, e \Downarrow M', c}{M, z \in e \Downarrow M', \mathit{belongs}\ z\ c} \qquad \texttt{union}\ \frac{M, e_1 \Downarrow M', c_1 \qquad M', e_2 \Downarrow M'', c_2}{M, e_1 \cup e_2 \Downarrow M'', \mathit{union}\ c_1\ c_2}$$

$$\texttt{intersection}\ \frac{M, e_1 \Downarrow M', c_1 \qquad M', e_2 \Downarrow M'', c_2}{M, e_1 \cap e_2 \Downarrow M'', \mathit{intersection}\ c_1\ c_2}$$

$$\texttt{difference}\ \frac{M, e_1 \Downarrow M', c_1 \qquad M', e_2 \Downarrow M'', c_2}{M, e_1 - e_2 \Downarrow M'', \mathit{difference}\ c_1\ c_2}$$

$$\texttt{included}\ \frac{M, e_1 \Downarrow M', c_1 \qquad M', e_2 \Downarrow M'', c_2}{M, e_1 \subseteq e_2 \Downarrow M'', \mathit{included}\ c_1\ c_2} \qquad \texttt{assign}\ \frac{M, e \Downarrow M', v}{M, x := e \Downarrow M' \prec\!+ (x, v), v}$$

Como se puede ver en las reglas de evaluación aparecen funciones auxiliares que serán de utilidad para trabajar con los conjuntos representados como listas.
Se deberán definir entonces las funciones:

- `belongs`, que dado un entero `z` y un conjunto `c`, retorna `True` si `z` se encuentra en `c`.
- `union`, que dados 2 conjuntos c_1 y c_2, retorna un nuevo conjunto con los elementos de ambos conjuntos.
- `intersection`, que dados 2 conjuntos c_1 y c_2, retorna un nuevo conjunto con los elementos que c_1 y c_2 tienen en común.
- `difference`, que dados 2 conjuntos c_1 y c_2, retorna un nuevo conjunto con los elementos de c_1, que no se encuentren en c_2.
- `included`, que dados 2 conjuntos c_1 y c_2, retorna `True` si todos los elementos de c_1 se encuentran en c_2.