UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Autor: Adrián Aguilera Moreno

Autómatas y Lenguajes Formales

Tarea 4

Considera la exponenciación:

$$\begin{aligned} \mathrm{n}^0 &= 1 \\ \mathrm{n}^{\mathrm{m}+1} &= \mathrm{n}^{\mathrm{m}} \times \mathrm{n} \end{aligned}$$

1. Transfórmala en una afirmación recursiva μ.

   Solución: Antes de definir la función recursiva-μ requerida, definamos una función recursiva-μ que defina a la siguiente función

   $$\begin{aligned} \mathrm{h}(0) &= \mathrm{s}(0) \\ \mathrm{h}(\mathrm{s}(\mathrm{n})) &= \mathrm{h}(\mathrm{n}) \end{aligned}$$

   como la función recursiva-μ

   $$\begin{aligned} \mathrm{h} : \mathbb{N} &\to \mathbb{N} \\ \mathrm{h}(\mathrm{cero}()) &= \mathrm{s} \circ \mathrm{cero}() \\ \mathrm{h}(\mathrm{s}(\mathrm{n})) &= \prod_{2}^{2}(\mathrm{n}, \mathrm{h}(\mathrm{n})). \end{aligned}$$

   Después de la definición anterior, encontremos una función recursiva-μ que cumpla con lo requerido[1], esta es

   > **Obs.** la función $\mathrm{g} : \mathbb{N}^3 \to \mathbb{N}$, se define por
   >
   > $$\mathrm{g} = \mathrm{mult} \circ \left(\prod_{1}^{3}, \prod_{3}^{3} \right)$$
   >
   > que no resulta tan relevante hacerla explicita en la definición de expt.

   $$\begin{aligned} \mathrm{expt} : \mathbb{N}^2 &\to \mathbb{N} \\ \mathrm{expt}(\mathrm{n}, 0) &= \mathrm{h}(\mathrm{m}) \\ \mathrm{expt}(\mathrm{n}, \mathrm{s}(\mathrm{m})) &= \left(\mathrm{mult} \circ \left(\prod_{1}^{3}, \prod_{3}^{3} \right) \right) (\mathrm{n}, \mathrm{m}, \mathrm{expt}(\mathrm{n}, \mathrm{m})). \end{aligned}$$

   □

2. A partir de la respuesta a 1, escríbela de acuerdo con las convenciones para el cálculo Λ.

   Solución: Para definir una función-λ que transcriba el funcionamiento de la función recursiva-μ presentada con anterioridad necesitamos definir al numeral 1, esto es

   $$\overline{1} \equiv_{def} \left[F, \overline{0}\right].$$

   Ahora, definimos la función-λ requerida[2], esto es

   $$\mathrm{expt} \equiv_{def} \mathrm{Y}\left(\lambda_{\mathrm{f}}.\lambda_{\mathrm{nm}}(\mathrm{Cero\ m})\ \overline{1}\ (*\ (\mathrm{P\ m})\ \mathrm{n}\ (\mathrm{f}((\mathrm{P\ m})\ \mathrm{n})))\right).$$

   debe quedar claro que los terminos n, m son la base y el exponente respectivamente. □

[1] Para esto haremos uso de la función `sum` y `mult` vistas en clase.
[2] Tomando en cuenta las funciones-λ $*, \mathrm{P}, \mathrm{Cero}$, y Y vistas en clase.

3. A partir de la respuesta a 1, impleméntala por medio de un programa de IMP.

Solución: Para esta problema, demos las siguientes definiciones recursivas en IMP[3]

$$\begin{aligned} \text{HEXPT} &= \text{X} := 1 \\ \text{GEXPT} &= \text{X} := \text{X} * \text{Z} \end{aligned}$$

a partir de lo anterior definimos[4]

1: EXPT

Input: N, M como respectivos en base y potencia.
Output: N^{M}.

```
1 Y := 0;
2 Z := N;
3 W := M;
4 HEXPT;
5 while Y < W do
6 |   GEXPT;
7 |   N := Z;
8 |   Y := Y + 1;
9 end
```

Comentarios por línea

1. Está línea indica nuestro contador.
2. Aquí recibimos la base de la potencia.
3. Recibimos el exponente de la potencia.
4. Llammamos lo equivalente a nuestro caso base (notar que no se esta haciendo de manera recursiva).
5. Verificamos la condición del `while`.
6. Llamamos a la función auxiliar que realiza la multiplicación de terminos.
7. Como los comandos resultan ser destructivos, recuperamos el valos de N (se hace la presición de el por qué no se recupera el valor de M. Pues no se utiliza en algún comando destructivo).
8. Se aumenta el contador en 1.
9. Termina el ciclo (esto pasa cuando la condición de validación en 5 ya no funciona, devuelve F). □

4. Define la semántica del comando

```
for i := 1 to n do P
```

y utilízalo para definir la recursión primitiva de una manera más directa.

Solución: Para este ejercicio supondremos la regla semántica que define al orden "<"[5], así podemos definir

$$\frac{\langle \text{i} := 1, \sigma \rangle \to \sigma' \qquad \langle \text{i} < \text{n}, \sigma' \rangle \to F}{\langle \texttt{for i := 1 to n do P}, \sigma \rangle \to \sigma}$$

[3] Bajo el supuesto de que $* :=$ MULT $\in$ IMP.
[4] Y suponiendo $+ :=$ SUMA $\in$ IMP.
[5] En general supondremos todas las definiciones dadas en clase, pero especificaremos como llamarle para hacerlas más compactas.

Nótese que el `for` se definió bajo el orden "<", sin embargo se puede construir una versión que admita el orden "≤" de la siguiente manera

$$\frac{\langle \mathrm{i} := 1, \sigma\rangle \to \sigma' \quad \langle \mathrm{i} < \mathrm{n} \ \vee \ \mathrm{i} = \mathrm{n}, \sigma'\rangle \to F}{\langle \texttt{for i := 1 to n do P}, \sigma\rangle \to \sigma}$$

Para este ejercicio particular tomaré la primer definición y se debe observar que es el equivalente a "h" en nuestras funciones recursivas-μ. Basados en esta definición, definimos

$$\frac{\langle \mathrm{i} := 1, \sigma\rangle \to \sigma'' \quad \langle \mathrm{i} < \mathrm{n}, \sigma''\rangle \to V \quad \langle P, \sigma''\rangle \to \sigma''' \quad \langle \texttt{for i := 1 to n do P}, \sigma'''\rangle \to \sigma'}{\langle \texttt{for i := 1 to n do P}, \sigma\rangle \to \sigma'_{[i:=i+1]}}$$

Ahora que tenemos la semántica del comando `for`, podemos definir la recursión primitiva basada en este comando.

> **Obs.** Para la definición de recursión primitiva se supone que existen P_h y P_g, pues sabemos que existen las funciones h y g como funciones recursivas-μ y por la demostración de la equivalencia entre funciones recursivas-μ y SOE, podemos concluir que sus homólogos [P_h y P_g] en SOE existen.

de la lámina[6] 30/32 tenemos que

$$\begin{aligned} \sigma'(\mathrm{X}_0) &= \mathrm{h}(\sigma(\mathrm{X})) && \text{donde } \langle P_h, \sigma\rangle \to \sigma' \\ \sigma'(\mathrm{X}_0) &= \mathrm{g}(\sigma(\mathrm{Z}), \sigma(\mathrm{X}), \sigma(\mathrm{X}')) && \text{donde } \langle P_g, \sigma\rangle \to \sigma' \end{aligned}$$

así, definimos el siguiente programa

2: Recursión Primitiva.

```
1 W := Z;
2 Y := X;
3 Ph;
4 for i := 1 to W do
5 |   Z := i - 1;
6 |   X := Y;
7 |   X' := X0;
8 |   Pg;
9 end
```

Obsérvese que este programa recorre a $\mathrm{i} \in \{1, \cdots, \mathrm{W} - 1\}$, y Z va tomando valores desde

$$0 \leq \mathrm{Z} \leq \mathrm{W} - 2$$

es por esto que se debe aceptar un valor para Z (que eventualmente se guardará en W) del valor requerido más uno, esto es, si se quisiera Z. Entonces llamamos al programa con $\mathrm{Z} + 1$, *i.e.*,

$$\mathrm{W} := \mathrm{Z} + 1$$

será el tope. Así, habrá Z[7] iteraciones[8]. Caso análogo si utilizaramos la definición alternativa para el for [la que considera a la igualdad]. □

[6] Otros modelos computacionales.
[7] W después de la primera iteración, pues aquí se conserva el valor inicial de Z.
[8] En el while de la lámina 30/32 justamente se itera de 0 a W - 1 veces, esto es W iteraciones.