Especificación de costos

Estructuras de Datos y Algoritmos II
Trabajo Práctico 2

Marzorati Denise
marzorati.denise@gmail.com

Soncini Nicolás
soncininicolas@gmail.com

8 de Junio de 2016

Docentes de la materia

Mauro Jaskelioff
Cecila Manzino
Juan M. Rabasedas
Martin Ceresa

Implementación con Listas

filterS

La implementación de $filterS$ con listas hace uso del paralelismo para reducir su profundidad lo mas posible en base a las aplicaciones de la función que se le pasa.

El `trabajo` de la función se puede tomar de la siguiente forma:

$$W\left(filterS\ f\ xs\right) \in O\left(\sum_{i=0}^{|xs|-1} W\left(f\ xs_i\right)\right)$$

De esta forma, se calcula la aplicacion de la función argumento en cada llamada recursiva de $filterS$ hasta que la lista termine vacía. En el mejor de los casos, si $W\left(f\ xs_i\right) \in O\left(1\right)$ queda su sumatoria como $|xs|$, por lo tanto se puede omitir sumar este valor en la cota superior del trabajo.

La `profundidad` de la función se puede reducir gracias a la aplicación en paralelo de la función argumento, por lo tanto se puede deducir que:

$$S\left(filterS\ f\ xs\right) \in O\left(|xs| + \max_{i=0}^{|xs|-1} S\left(f\ xs_i\right)\right)$$

En este caso, como el máximo de todos los trabajos puede quedar constante, debemos sumar el trabajo de obtener cada elemento, lo cual suma $|xs|$.

showtS

La implementación de $showtS$ con listas divide la lista en dos mitades y con las funciones $takeS$ y $dropS$.

Sabemos que las funciones utilizadas son lineales:

$$W/S\left(takeS\ xs\ n\right) \in O\left(|xs|\right)$$
$$W/S\left(dropS\ xs\ n\right) \in O\left(|xs|\right)$$

tanto en su trabajo como en su profundidad.

Luego, podemos concluir fácilmente que valen los siguientes costos ya que se aplica una vez cada función sobre la mitad de la lista.

$$W\left(showtS\ xs\right) \in O\left(|xs|\right)$$

$$S\left(showtS\ xs\right) \in O\left(|xs|\right)$$

reduceS

La implementación de $reduceS$ toma una función de costo desconocido, un elemento y una lista del mismo tipo y aplica el algoritmo de reduce de forma que opera con la función dada $\oplus$ los elementos de la lista de a pares sobre el resultado de cada llamada recursiva. Este orden de reducción (u operación) es el dado por el TAD.

Su `trabajo` y profundidad se pueden calcular teniendo en cuenta que la implementación del algoritmo emplea una reducción del tamaño del arreglo en las llamadas a la función $contraer$, que cumple:

$$W\left(contraer\ \oplus\ xs\right) \in O\left(\sum_{i=0}^{(|xs|/2)-1} W\left(xs_{2*i} \oplus xs_{(2*i)+1}\right)\right)$$

$$S\,(contraer\ \oplus\ xs) \in O\left(|xs| + \max_{i=0}^{(|xs|/2)-1} S\left(xs_{2*i} \oplus xs_{(2*i)+1}\right)\right)$$

Ahora, a la hora de calcularlos para *reduceS*, nosotros debemos forzar el orden de reducción, ya que la implementación de *contraer* utiliza un orden que no cumple la especificación necesaria, pero es fácil identificar el rol que cumple al ser bien integrado a *reduceS*. Como primera función auxiliar se encuentra *contraer*, que dada una función y una lista, opera de a pares sus elementos, tomando las llamadas recursivas y las aplicaciones de la función en forma paralela. Debemos entonces calcular los costos de *contraer* para poder calcular los de la función pedida:

$$W\,(contraer\ \oplus\ xs) \in O\left(|xs| + \sum_{i=0}^{(|xs/2|)-1} W\left(xs_{2i} \oplus xs_{2i+1}\right)\right)$$

$$S\,(contraer\ \oplus\ xs) \in O\left(|xs| + \max_{i=0}^{(|xs/2|)-1} S\left(xs_{2i} \oplus xs_{2i+1}\right)\right)$$

Es claro entonces el costo que aporta adecuar esta función para que trabaje sobre cada resultado de un llamado recursivo de si mismo, cumpliendo así el orden de reducción pedido por el TAD de Secuencias para *reduceS*. Ahora si podemos calcular el trabajo de *reduceS*, el cual queda de la siguiente manera:

$$W\,(reduceS \oplus\ b\ xs) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, b, xs)} W\left(xs_i \oplus xs_j\right)\right)$$

Tanto su `trabajo` como su `profundidad` se ven afectadas por el reordenamiento de reducción sobre el resultado de *contraer*, con lo cual la paralelización de ésta no puede ser aprovechada, y obtenemos:

$$S\,(reduceS \oplus\ b\ xs) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, b, xs)} S\left(xs_i \oplus xs_j\right)\right)$$

`scanS`

La implementación de `scanS` hace uso de varias funciones auxiliares para su correcto funcionamiento y su fácil comprensión. Para obtener el costo del mismo debemos primero describir y especificar los costos de las funciones que lo auxilian.

Como primera función auxiliar se encuentra *contraer*, que dada una función y una lista, opera de a pares sus elementos, tomando las llamadas recursivas y las aplicaciones de la función en forma paralela. (Se detallan los costos en la especificación de costos de reduceS).

Luego hacemos uso de la función *expandir*, que dada una función y dos listas, devuelve una lista.

$$W\,(expandir\ \oplus\ xs\ zs) \in O\left(|xs| + \max_{i=0}^{(|xs/2|)-1} W\left(xs_{2i} \oplus xs_{2i+1}\right)\right)$$

$$S\,(expandir\ \oplus\ xs\ zs) \in O\left(|xs| + \max_{i=0}^{(|xs/2|)-1} S\left(xs_{2i} \oplus xs_{2i+1}\right)\right)$$

Podemos concluír de esta forma, dado que la función *scanS* para listas realiza llamados de expandir sobre su resultado recursivo al aplicar contraer a la lista dada, que su `trabajo` es por lo menos lineal, el se calcula como:

$$W\,(scanS \oplus\ b\ xs) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, b, xs)} W\left(xs_i \oplus xs_j\right)\right)$$

Y dado que las profundidades de *expandir* y *contraer* son como mínimo lineales, la `profundidad` de *scanS* queda como la suma del tamaño de la lista y el máximo de las profundidades de la aplicación de la función sobre las sub-aplicaciones en el orden de reducción dado. Con lo cual tenemos:

$$S\,(scanS \oplus\; b\; xs) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, b, xs)} S\,(xs_i \oplus xs_j)\right)$$

Implementación con Arreglos Persistentes

filterS

La función $filterS$ toma una lista y una función sobre elementos de la misma y hace uso de *tabulateS* para aplicar la función sobre cada elemento y $joinS$ para unir los resultados sueltos de la aplicación anterior. Sabemos que los costos de *tabulateS* y $joinS$ para arreglos persistentes son:

$$W\,(\mathit{tabulateS}\ f\ n) \in O\left(\sum_{i=0}^{n-1} W\,(f\ i)\right)$$

$$S\,(\mathit{tabulateS}\ f\ n) \in O\left(\max_{i=0}^{n-1} S\,(f\ i)\right)$$

y

$$W\,(joinS\ as) \in O\left(|as| + \sum_{i=0}^{|as|-1} (|as_i|)\right)$$

$$S\,(joinS\ \ as) \in O\,(\ lg\ |as|\ )$$

De ambas se puede calcular, TODOTODOTODOTODOTODOTODO

$$W\,(filterS\ f\ as) \in O\left(\sum_{i=0}^{|as|-1} W\,(f\ as_i)\right)$$

$$S\,(filterS\ f\ as) \in O\left(\ lg\ |as| + \max_{i=0}^{|as|-1} S\,(f\ as_i)\right)$$

showtS

Para la implementación de *showtS* para este formato debemos partir al arreglo en dos partes de forma que nos de una rapida idea de la forma en árbol del mismo. Para llevar a cabo esto hacemos uso de las funciones ya definidas para arreglos persistentes *takeS* y *dropS*, que devuelven un arreglo con los primeros o últimos elementos del arreglo según una cantidad arbitraria respectivamente. Para considerar los costos de *showtS* sabemos como primera instancia que los costos de las funciones auxiliares son constantes:

$$W/S\,(\mathit{takeS}\ as\ n) \in O\,(|1|)$$

$$W/S\,(\mathit{dropS}\ as\ n) \in O\,(|1|)$$

Y como tomar su longitud también lo es (útil para el cálculo de $2^{ilog(|as|-1)}$), tenemos entonces que los costos para la función *showtS* también quedan constantes:

$$W\,(\mathit{showtS}\ as) \in O\,(1)$$

$$S\,(\mathit{showtS}\ as) \in O\,(1)$$

`reduceS`

$$W\left(contraer\ \oplus\ xs\right) \in O\left(\sum_{i=0}^{(|xs|/2)-1} W\left(xs_{2*i} \oplus xs_{(2*i)+1}\right)\right)$$

$$S\left(contraer\ \oplus\ xs\right) \in O\left(|xs| + \max_{i=0}^{(|xs|/2)-1} S\left(xs_{2*i} \oplus xs_{(2*i)+1}\right)\right)$$

$$W\left(contraer\ \oplus\ xs\right) \in O\left(|xs| + \sum_{i=0}^{(|xs/2|)-1} W\left(xs_{2i} \oplus xs_{2i+1}\right)\right)$$

$$S\left(contraer\ \oplus\ xs\right) \in O\left(|xs| + \max_{i=0}^{(|xs/2|)-1} S\left(xs_{2i} \oplus xs_{2i+1}\right)\right)$$

$$W\left(reduceS \oplus\ b\ xs\right) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, b, xs)} W\left(xs_i \oplus xs_j\right)\right)$$

$$S\left(reduceS \oplus\ b\ xs\right) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, b, xs)} S\left(xs_i \oplus xs_j\right)\right)$$

`scanS`

$$W\left(expandir\ \oplus\ xs\ zs\right) \in O\left(|xs| + \max_{i=0}^{(|xs/2|)-1} W\left(xs_{2i} \oplus xs_{2i+1}\right)\right)$$

$$S\left(expandir\ \oplus\ xs\ zs\right) \in O\left(|xs| + \max_{i=0}^{(|xs/2|)-1} S\left(xs_{2i} \oplus xs_{2i+1}\right)\right)$$

$$W\left(scanS \oplus\ b\ xs\right) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, b, xs)} W\left(xs_i \oplus xs_j\right)\right)$$

$$S\left(scanS \oplus\ b\ xs\right) \in O\left(|xs| + \sum_{(xs_i \oplus xs_j) \in \mathcal{O}_r(\oplus, b, xs)} S\left(xs_i \oplus xs_j\right)\right)$$