UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Tarea 2

INTEGRANTES

Torres Valencia Kevin Jair - 318331818
Aguilera Moreno Adrián - 421005200
Rivera Silva Marco Antonio - 318183583

PROFESORA

Karla Ramírez Pulido

AYUDANTES

Alan Alexis Martínez López
Manuel Ignacio Castillo López
Alejandra Cervera Taboada

ASIGNATURA

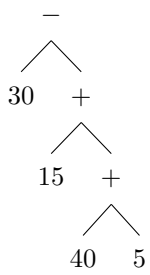
Lenguajes de Programación

12 de septiembre de 2022

1. Dadas las expresiones de `WAE` en sintaxis concreta, da su respectiva representación en sintaxis abstracta por medio de los Arboles de Sintaxis Abstracta correspondientes. En caso de no poder generar el árbol, justifica.

a) $\{- \ 30 \ \{+ \ 15 \ \{+ \ 40 \ 5 \ \} \ \} \ \}$

La representación del `ASA` para la expresión anterior, tiene la forma siguiente

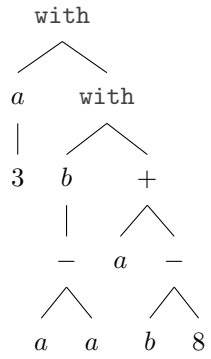


b) $\{- \ 30 \ \{+ \ 300 \ \{+ \ 40 \ \} \ \} \ \}$

En este caso, particular, las operaciones[1] no tienen sumandos y minuendo izquierdo. Por tanto, no podemos construir el `ASA`, pues sería un árbol incompleto (la expresión en sintaxis concreta es incompleta).

c) $\{$ `with` $\{a \ 3\}$
$\{$ `with` $\{b \ \{- \ a \ a\} \}$
$\{+ \ a \ \{- \ b \ \ 8 \ \} \ \}$

La representación del `ASA` para la anterior expresión es

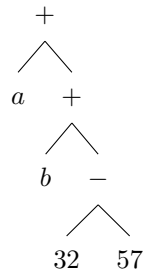


2. Dadas las siguientes expresiones de `WAE` en sintaxis concreta, obtén su sintaxis abstracta correspondiente y realiza la sustitución que se indica.

a) $e = \{+ a \ \{+ b \ \{- 32 \ \ 57 \} \} \}$

```
(subst (parse e) 'a (add (num 3) (num 4)))
```

Primero demos la representación abstracta de la expresión anterior, esta es



[1] 2 sumas y una resta.

Luego, apliquemos la instrucción `parse` a la expresión `e`, esto es

$$\begin{aligned}
\texttt{(parse e)} &= \texttt{(add (parse(}a\texttt{)) (parse(\{+ }b\texttt{ \{- 35 57\}\})))} \\
&= \texttt{(add (id '}a\texttt{) (add (parse(}b\texttt{)) (parse(\{- 35 57\}))))} \\
&= \texttt{(add (id '}a\texttt{) (add (id '}b\texttt{) (sub (parse(35)) (parse(57)))))} \\
&= \texttt{(add (id '}a\texttt{) (add (id '}b\texttt{) (sub (num 35) (num 57))))} \\
&= \texttt{e'}
\end{aligned}$$

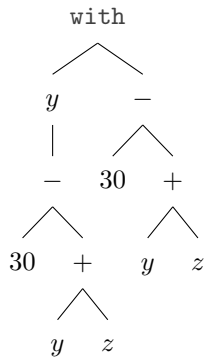
Ahora, apliquemos la instrucción `subst` a `e'` y los valores indicados, esto es

> **Obs.** Por simplicidad, asumimos que
>
> `subst(e') = (subst (e') 'a (add (num 3) (num 4)))`

$$\begin{aligned}
\texttt{subst (e')} &= \texttt{(add (subst (id '}a\texttt{)) (subst ((add (id '}b\texttt{) (sub (num 35) (num 57))))))} \\
&= \texttt{(add (add (num 3) (num 4)) ((add (subst (id '}b\texttt{)) (subst (sub (num 35) (num 57))))))} \\
&= \texttt{(add (add (num 3) (num 4)) ((add (id '}b\texttt{) (sub (subst (num 35)) (subst (num 57))))))} \\
&= \texttt{(add (add (num 3) (num 4)) ((add (id '}b\texttt{) (sub (num 35) (num 57))))).}
\end{aligned}$$

b) $e =$ `{ with {`y `{-` 30 `{-` y z `} }`
`{-` 30 `{+` y z `} }`
`(subst (parse e) 'y (id 'w))`

Primero veamos como esta construida la representación abstracta de la expresión anterior, esta es



Luego, apliquemos la instrucción `parse` a la expresión `e`, esto es

$$\begin{aligned}
\texttt{(parse e)} &= \texttt{(with (parse \{ }y\texttt{ \{- 30 \{- }y\ z\texttt{ \}\}) (parse \{- 30 \{+ }y\ z\texttt{ \}\}))} \\
&= \texttt{(with (((parse }y\texttt{) (parse \{- 30 \{- }y\ z\texttt{ \}\})))))} \\
&\quad \texttt{(sub (parse 30) (parse \{+ }y\ z\texttt{ \})))} \\
&= \texttt{(with ((id '}y\texttt{) (sub (parse 30) (parse \{- }y\ z\texttt{ \}\}))))} \\
&\quad \texttt{(sub (num 30) (add (parse }y\texttt{) (parse }z\texttt{))))} \\
&= \texttt{(with ((id '}y\texttt{) (sub (num 30) (sub (parse }y\texttt{) (parse }z\texttt{)))))} \\
&= \texttt{(sub (num 30) (add (id '}y\texttt{) (id '}z\texttt{))))} \\
&= \texttt{(with ((id '}y\texttt{) (sub (num 30) (sub (id '}y\texttt{) (id '}z\texttt{)))))} \\
&= \texttt{(sub (num 30) (add (id '}y\texttt{) (id '}z\texttt{))))} \\
&= \texttt{e'}
\end{aligned}$$

Ahora, apliquemos la instrucción `subst` a `e'` y los valores indicados, esto es (se obvia la observación anterior)

```
(subst e)  =  (subst (with ((id 'y) (sub (num 30) (sub (id 'y) (id 'z)))))
              (sub (num 30) (add (id 'y) (id 'z)))))
           =  (with ((id 'y) (subst (sub (num 30) (sub (id 'y) (id 'z))))))
              (sub (num 30) (add (id 'y) (id 'z)))))
           =  (with ((id 'y) (sub (subst (num 30)) (subst (sub (id 'y) (id 'z))))))
              (sub (num 30) (add (id 'y) (id 'z)))))
           =  (with ((id 'y) (sub (num 30) (sub (subst (id 'y)) (subst (id 'z))))))
              (sub (num 30) (add (id 'y) (id 'z)))))
           =  (with ((id 'y) (sub (num 30) (sub (id 'w) (id 'z)))))
              (sub (num 30) (add (id 'y) (id 'z)))))
```

.

c) e = { with{y {− 30 {− y z } }

{− 30 {+ y z } }

```
(subst (parse e) 'z (id 'v))
```

Observemos que la representación abstracta para este inciso es la misma que para el inciso anterior, pues la expresión `with`, en principio, es la misma. Así, omitimos este paso.

Aplicar la instrucción `parse` para este inciso nos devuelve lo mismo que en el inciso anterior, así omitimos este paso y tomaremos el valor de `e'` encontrado en el inciso previo.

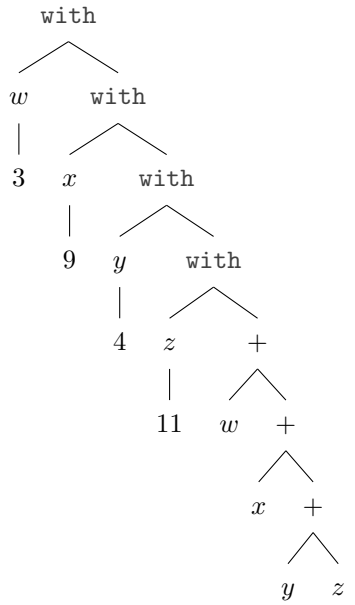
Ahora, apliquemos la instrucción `subst` a `e'` y los valores indicados, esto es (se obvia la observación hecha en este ejercicio)

```
(subst e)  =  (subst (with ((id 'y) (sub (num 30) (sub (id 'y) (id 'z)))))
              (sub (num 30) (add (id 'y) (id 'z)))))
           =  (with (subst ((id 'y) (sub (num 30) (sub (id 'y) (id 'z))))))
              (subst (sub (num 30) (add (id 'y) (id 'z)))))
           =  (with ((id 'y) (subst (sub (num 30) (sub (id 'y) (id 'z))))))
              (sub (subst (num 30)) (subst (add (id 'y) (id 'z)))))
           =  (with ((id 'y) (sub (subst (num 30)) (subst (sub (id 'y) (id 'z))))))
              (sub (num 30) (add (subst (id 'y)) (subst (id 'z))))))
           =  (with ((id 'y) (sub (num 30) (sub (subst (id 'y)) (subst (id 'z))))))
              (sub (num 30) (add (id 'y) (id 'v)))))
           =  (with ((id 'y) (sub (num 30) (sub (id 'y) (id 'v)))))
              (sub (num 30) (add (id 'y) (id 'v))))).
```

c3. Sea la siguiente expresión del lenguaje `WAE`. Da la sintaxis abstracta de la misma, muestra el proceso de evaluación mediante la función **interp** y responde las siguientes preguntas.

{ with { w 3 }
{ with{ x 9 }
{ with { y 4 }
{ with { z 11 }
{+ w {+ x {+ y z } } } } } } }

Veamos que la representación de la sintaxis abstracta de la expresión anterior es



Aplicando la función `interp` tenemos que

```
> {with {w 3}
        {with {x 9}
            {with {y 4}
                {with {z 11}
                    {+ w {+ x {+ y z}}}}}}}
= {with {x 9}
        {with {y 4}
              {with {z 11}
                    {add (num 3) {+ x {+ y z}}}}}}
= {with {y 4}
        {with {z 11}
              {add (num 3) {add (num 9) {+ y z}}}}}
= {with {z 11}
        {add (num 3) {add (num 9) {add (num 4) z}}}}
= {add (num 3) {add (num 9) {add (num 4) (num 11)}}}
= {add (num 3) {add (num 9) (num 15)}}
= {add (num 3) (num 24)}
= (num 27)
```

a) ¿Cuántas veces se aplica el algoritmo de sustitución para evaluar la expresión?

Para el `with` con `id = w` tenemos que el algoritmo de sustitución se empieza a aplicar desde su cuerpo, después se aplica al valor (`val`) y cuerpo del `with` con `id = x` (pues `w` $\neq$ `x`), esto pasa también, para el `with` con `id = y` e `id = z` (hasta aquí hemos aplicado el algoritmo 7 veces, sin aplicarlo al último cuerpo). Eventualmente, llegamos al cuerpo de `with` con `id = z` donde aplicamos el agoritmo de sustitución 6 veces más.

A lo anterior le tenemos que sumar las aplicaciones del algoritmo de sustitución si comenzamos desde el `with` con `id = x`, que son 5 veces más las 6 aplicaciones del último cuerpo.

También sumamos las aplicaciones si iniciamos desde el `with` con `id = y`, que son 3 más las 6 del último cuerpo.

Por último contamos las aplicaciones con el `with` de `id = z`, estas son 1 más las 6 dentro de su cuerpo.

Así, tenemos que en total aplicamos 40 veces el algoritmo de sustitución (esto tomando en cuenta cada llamada recursiva). El algoritmo se aplica para 4 variables en total (que están en el último cuerpo).

b) ¿Qué pasaría si añadimos, $100,000$ sumas con $100,000$ identificadores nuevos a la expresión? ¿Cuántas sustituciones se tendrían que hacer? ¿Qué nos dice esto con respecto al rendimiento de la función **interp**?

Recordemos que el algoritmo de sustitución es de orden $\mathcal{O}(n^2)$, así la cota será de aproximadamente $(100,004)^2$ aplicaciones en el algoritmo de sustitución.

La función `interp` es ineficiente para cantidades de identificadores grandes.

4. Convierte las siguientes expresiones a su respectiva versión usando índices de De Bruijn.

```
{with {v 2}
    {with {w 3}
        {with {x 4}
            {with {y {+ v {- w x} } }
                {with {z {with {v {+ w x } } v } }
                    {+ y {with {w {- y z } } {- w x } } } } } } } } }
```

Versión con indices de Bruijn

```
{with 2
    {with 3
        {with 4
            {with {+ <: 2 0> {- <: 1 0> <: 0 0> } }
                {with {with {+ <: 2 0> <: 1 0>} <: 3 0> }
                    {+ <: 1 0> {with {- <: 1 0> <: 0 0> } {- <: 3 0> <: 2 0> }}}}}}}}}
```

5. Dadas las siguientes expresiones representadas mediante índices de De Bruijin, obtén su respectiva versión usando identificadores de variables.

```
{with {1 2 3}
  {with {4 5 6}
    {with { {with { { + <:0 1> <:1 2>} {−<:1 1> <:0 0>} }3}}
      {with {<:0 0>}
        {+ <:3 2>{+ <:2 1> {+ <:1 0> <:0 0>}}}}}}}
```

Versión usando identificadores de variables:

```
{with {{a 1}  {b 2} {c 3}}
  {with {{d 4} {e 5} {f 6}}
    {with { {g {with { {h {+ e c}} {i{− b d} }3}}}
      {with {j g}
        {+ c{+ e {+ g j}}}}}}}}
```

6. Dada la siguiente expresión.

```
{with {w 2}
    {with {x 3}
        {with {y {+ w x } }
            {with { w −2}
                {with {x −3 }
                    {+ y y } } } } } }
```

a. Determinar el valor de la expresión. Muestra los pasos que hiciste para hacerlo. El proceso es el siguiente.

Empezaremos con el primer with, podemos ver que la variable w y se le asignará el valor de 2.

```
{with {w 2}
```

Ahora pasamos al cuerpo del primer with. podemos ver que la variable x todas las veces que salga en el cuerpo de este segundo with será sustituida por el valor de 3.

```
{with {w 2}
    {with {x 3}
```

Ahora pasamos al cuerpo del segundo with (que es otro with). Podemos ver que la variable y va a tener el valor de $+wx$, pero los anteriores with ya nos dieron estos valores, por lo que nuestro valor para y es {+ 2 3}.

```
{with {w 2}
    {with {x 3}
        {with {y {+ w x } }
```

Ahora pasamos al cuerpo del tercer with (que es otro with). aquí se nos vuelve a mostrar la variable w sin embargo, todos los with dentro de este with tendrán en valor de la w que se defina en este with, ya que opaca a las anteriores asignaciones. Por lo que el valor para esta w es -2.

```
{with {w 2}
    {with {x 3}
        {with {y {+ w x } }
            {with { w −2}
```

Pasamos al cuerpo del cuarto with, que también es un with. Podemos ver que se vuelve a aplicar un valor a x, sin embargo sólo será tomado en cuenta para withs anidados dentro de este with. El valor para la x es -2.

```
{with {w 2}
    {with {x 3}
        {with {y {+ w x } }
            {with { w −2}
                {with {x −3 }
```

Finalmente llegamos al cuerpo del 5to with (el cual ya no es un with xd). Podemos ver que devuelve la operación {+ y y}. Sin embargo anteriormente ya habiamos obtenido el valor de y, el cual era {+ 2 3}. por lo que lo sustituimos en {+ y y} quedandonos {+ {+ 2 3}{+ 2 3}}.

```
{with {w 2}
    {with {x 3}
        {with {y {+ w x } }
            {with { w -2}
                {with {x -3 }
                    {+ y y } } } } } }
```

Ahora evaluamos la operación resultante {+ {+ 2 3}{+ 2 3}} = {+ 5 5} = 10

b. ¿Pueden haber otros resultados? ¿Por qué?

Sí, pueden haber otros resultados, esto debido a que depende del tipo de alcance que tienen los withs ya que existen 2 tipos, dinámico y estático. Dependiendo de cual se use, las variables se pueden buscar desde arriba, o el más cercano hacia abajo lo cual puede dar un resultado diferente en caso de que coloquemos id's iguales.

c. ¿Cuál es el resultado correcto en dado caso de haber más de un posible resultado?

Depende de la implementación de la función y del lenguaje de programación, pues si hay más de un resultado correcto es porque el lenguaje lo permite lo cual hace responsable al programador sobre el uso que requiera, o en otras palabras, el resultado correcto será aquel contemplado por el programador para realizar los calculos que requiere.