TRABAJO FINAL - Extensión LIS

PROPUESTA

Voy a extender el lenguaje con:

➔ **Operador ternario**

intexp ::= boolexp ‘?’ intexp ‘:’ intexp

➔ **+= (Syntactic Sugar)**

comm ::= var ‘+=’ intexp ‘;’

➔ **-= (Syntactic Sugar)**

comm ::= var ‘-=’ intexp ‘;’

EXPLICACIÓN COLOQUIAL

➔ **Operador ternario**

El if de una línea o operador ternario, permite al usuario ejecutar 2 expresiones enteras según el valor booleano de entrada.

```
x:= 5

true ? x:=20 : x:=10

-- valor final de x = 20 por la condición true.
```

➔ **+=**

Azúcar sintáctico que reemplaza la expresión (x+=e) **≡** (x = x + e)

```
x:= 5

x += 5;  -- (x = x+5) valor final de x = 10
```

➔ **-=**

Azúcar sintáctico que reemplaza la expresión (x-=e) ≡ (x = x - e)

```
x:= 100
x -= 50;  -- (x = x - 50) valor final de x = 50
```

SINTAXIS CONCRETA

➔ **Operador ternario**

<intexp> ::= <boolexp> '?' <intexp> : <intexp>

➔ **+=**

<comm> ::= <var> '+=' <intexp> ';'

➔ **-=**

<comm> ::= <var> '-=' <intexp> ';'

SINTAXIS ABSTRACTA

➔ **Operador ternario**

```
data IntExp = ...
            | Question BoolExp IntExp IntExp
```

SEMÁNTICA

➔ **Operador ternario**

$$\frac{(e,\sigma) \Downarrow_{bool} true \quad (e1,\sigma) \Downarrow_{intexp} n}{(e\ ?\ e1\ :\ e2\ end, \sigma) \Downarrow_{intexp} n} \text{ if\_true}$$

$$\frac{(e,\sigma) \Downarrow_{bool} false \quad (e2,\sigma) \Downarrow_{intexp} m}{(e\ ?\ e1\ :\ e2\ end, \sigma) \Downarrow_{intexp} m} \text{ if\_false}$$

➔ **+=**

$$\frac{(e,\sigma) \Downarrow_{int} n \quad \sigma(v) = var \quad \sigma' = \sigma[v \mapsto var + n]}{(v\ +=\ e, \sigma) \Downarrow \sigma'}$$

➔ **-=**

$$\frac{(e,\sigma) \Downarrow_{int} n \quad \sigma(v) = var \quad \sigma' = \sigma[v \mapsto var - n]}{(v\ -=\ e, \sigma) \Downarrow \sigma'}$$

GRAMÁTICA INTEXP

intexp → intexp ‘+’ term | intexp ‘-’ term | term

term → term ‘*’ factor | term ‘/’ factor | factor

factor → ‘(‘ intexp ‘)’ | ‘-’ factor | var | id

GRAMÁTICA BOOLEXP

boolexp → boolexp ‘|’ boolexp2 | boolexp2

boolexp2 → boolexp2 ‘&’ boolexp3 | boolexp3

boolexp3 → ‘(’ boolexp ‘)’ | ‘~’ boolexp3 | intcomp | boolvalue

intcomp → intexp’ compopp intexp’

compopp → ‘=’ | ‘<’ | ‘>’

boolvalue → ‘true’ | ‘false’

GRAMÁTICA INTEXP + TERNARIO

intexp → boolexp ‘?’ intexp ‘:’ intexp | intexp’

intexp’ → intexp’ ‘+’ term | intexp’ ‘-’ term | term

term → term ‘*’ factor | term ‘/’ factor | factor

factor → ‘(‘ intexp ‘)’ | ‘-’ factor | var | id