2ª Aula Prática de Compiladores: Tree Walking com ANTLR 4

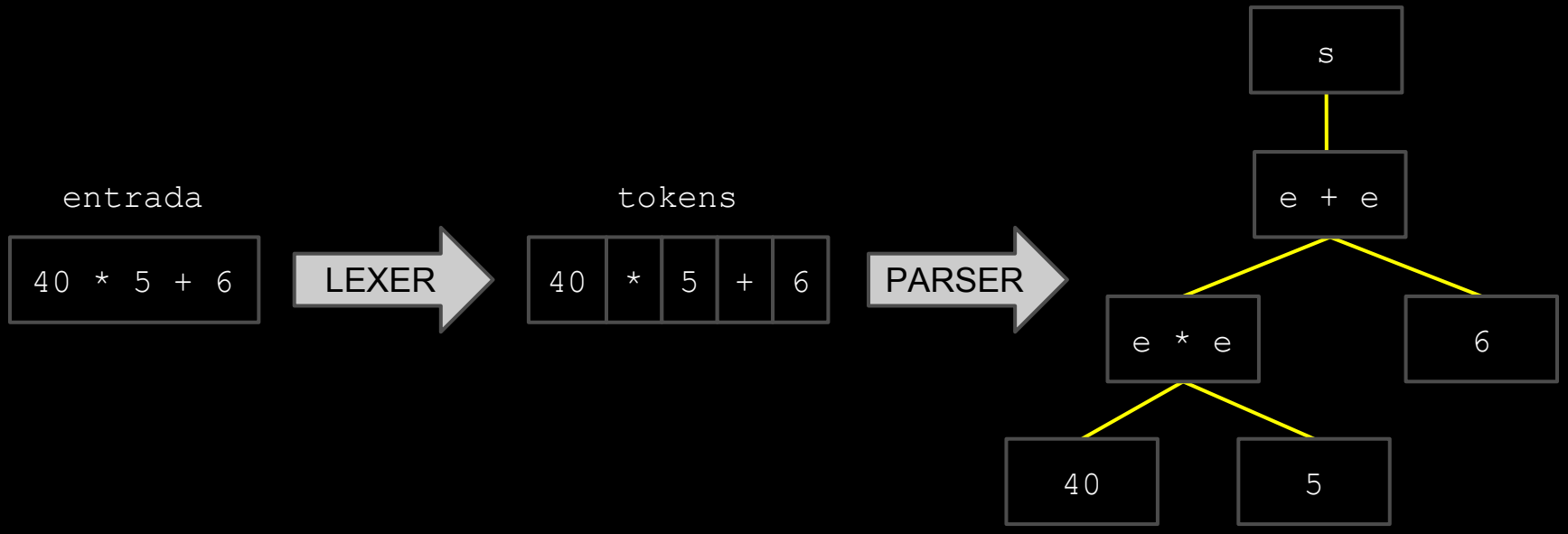
Monitoria de Teoria e Implementação de Linguagens Computacionais 2017.2

Roteiro

- Tree Walking
- Labels
- Listener
- Visitor
- Geração de Arquivos
- Exercícios Práticos

Tree Walking

- Procedimento de avaliação recursiva na árvore de derivação
- O ANTLR suporta dois mecanismos de tree-walking: listeners e visitors



Tree Walking (Listeners vs Visitors)

- Listeners
  - Geradas por padrão pelo ANTLR
  - Basta implementar os callbacks que o ANTLR irá chamá-los durante a exploração da árvore
- Visitors
  - Mais versáteis do que os listeners
  - É necessário implementar as chamadas aos nós filhos para visitar as sub-árvores

Labels

- O ANTLR gera métodos no listener e no visitor de acordo com as regras de produção da gramática
- Por padrão, não são criados métodos para as alternativas de cada regra
- Podemos "etiquetar" com uma label as alternativas das regras que desejamos que seja gerado um método no listener e no visitor

Labels (Gramática Exp vs Gramática AnnotExp)

```
grammar Exp;

s : e;
e : e '*' e
  | e '+' e
  | INT
  ;

INT : [0-9]+;
WS  : [ \t\r\n]+ -> skip;
```

Labels (Gramática Exp vs Gramática AnnotExp)

```
grammar AnnotExp;

s : e;
e : e MUL e     #MulExp
  | e ADD e     #AddExp
  | INT         #Int
  ;

MUL : '*';
ADD : '+';
INT : [0-9]+;
WS  : [ \t\r\n]+ -> skip;
```

Listener

- Se origina no padrão Observer, onde observadores (ou ouvintes) são notificados quando determinados eventos ocorrem
- São criados métodos de entrada (enter rule) e de saída (exit rule) para cada regra de produção e para as alternativas etiquetadas
- O ANTLR disponibiliza um objeto walker (no qual o listener é anexado) que pode ser utilizado também em outras etapas do processo de compilação

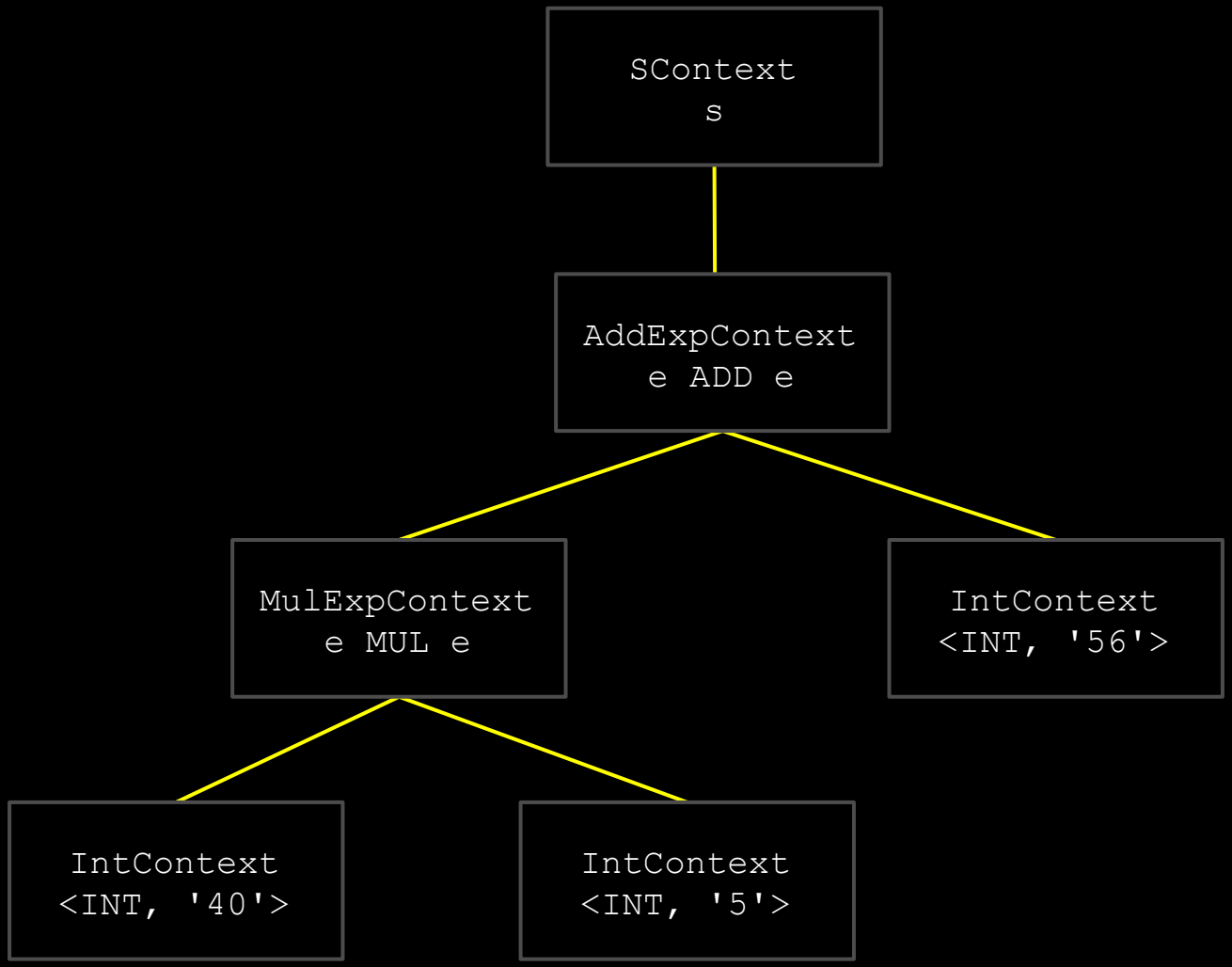
Listener (Exemplo de Interface de Listener no ANTLR)

```
public interface AnnotExpListener
extends ParseTreeListener {
    void enterS(AnnotExpParser.SContext ctx);
    void exitS(AnnotExpParser.SContext ctx);
    void enterMulExp(AnnotExpParser.MulExpContext ctx);
    void exitMulExp(AnnotExpParser.MulExpContext ctx);
    void enterAddExp(AnnotExpParser.AddExpContext ctx);
    void exitAddExp(AnnotExpParser.AddExpContext ctx);
    void enterInt(AnnotExpParser.IntContext ctx);
    void exitInt(AnnotExpParser.IntContext ctx);
}
```

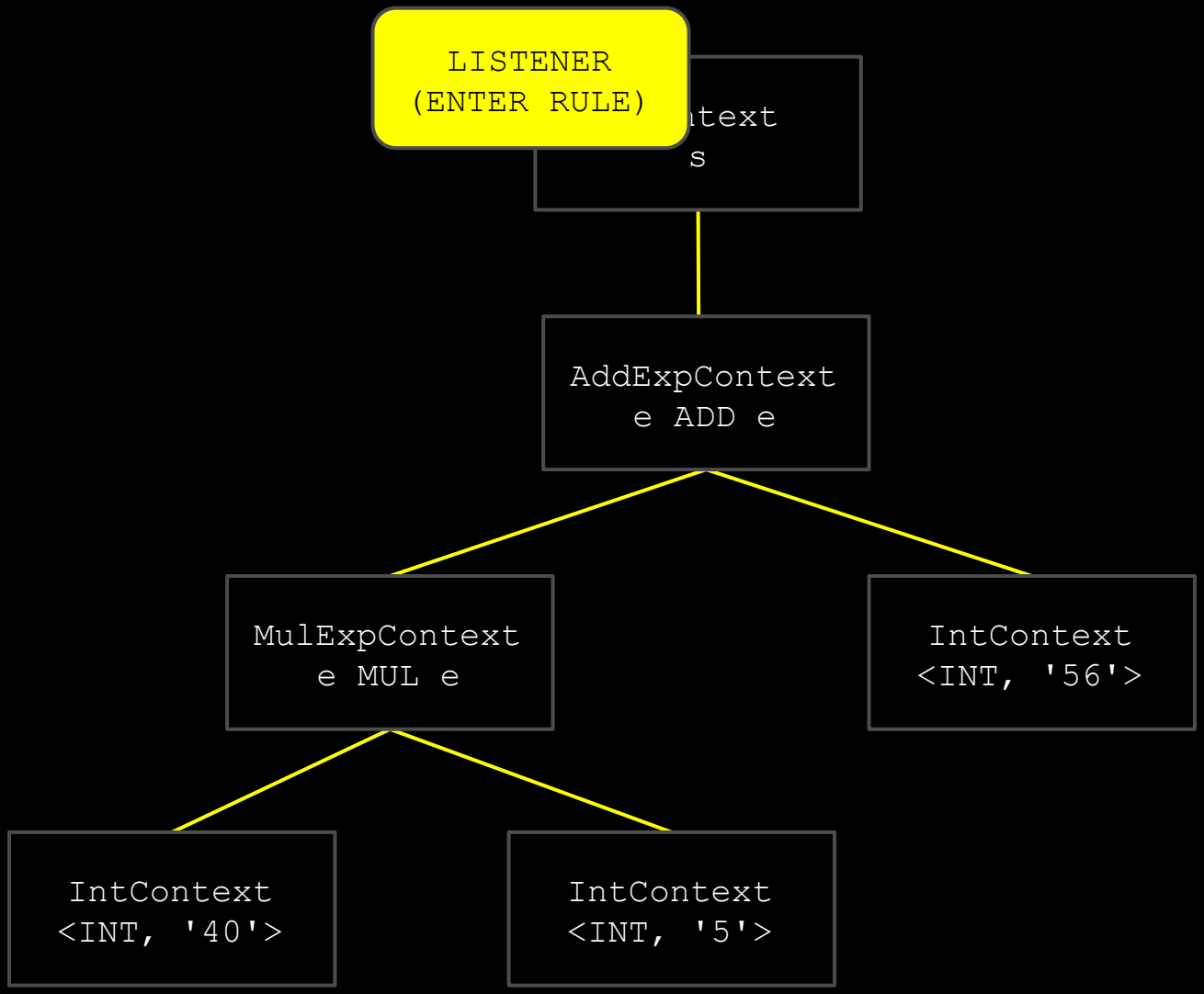
Listener (Exemplo)

- Percorrendo a árvore sintática gerada pela expressão “40 * 5 + 56” da gramática AnnotExp

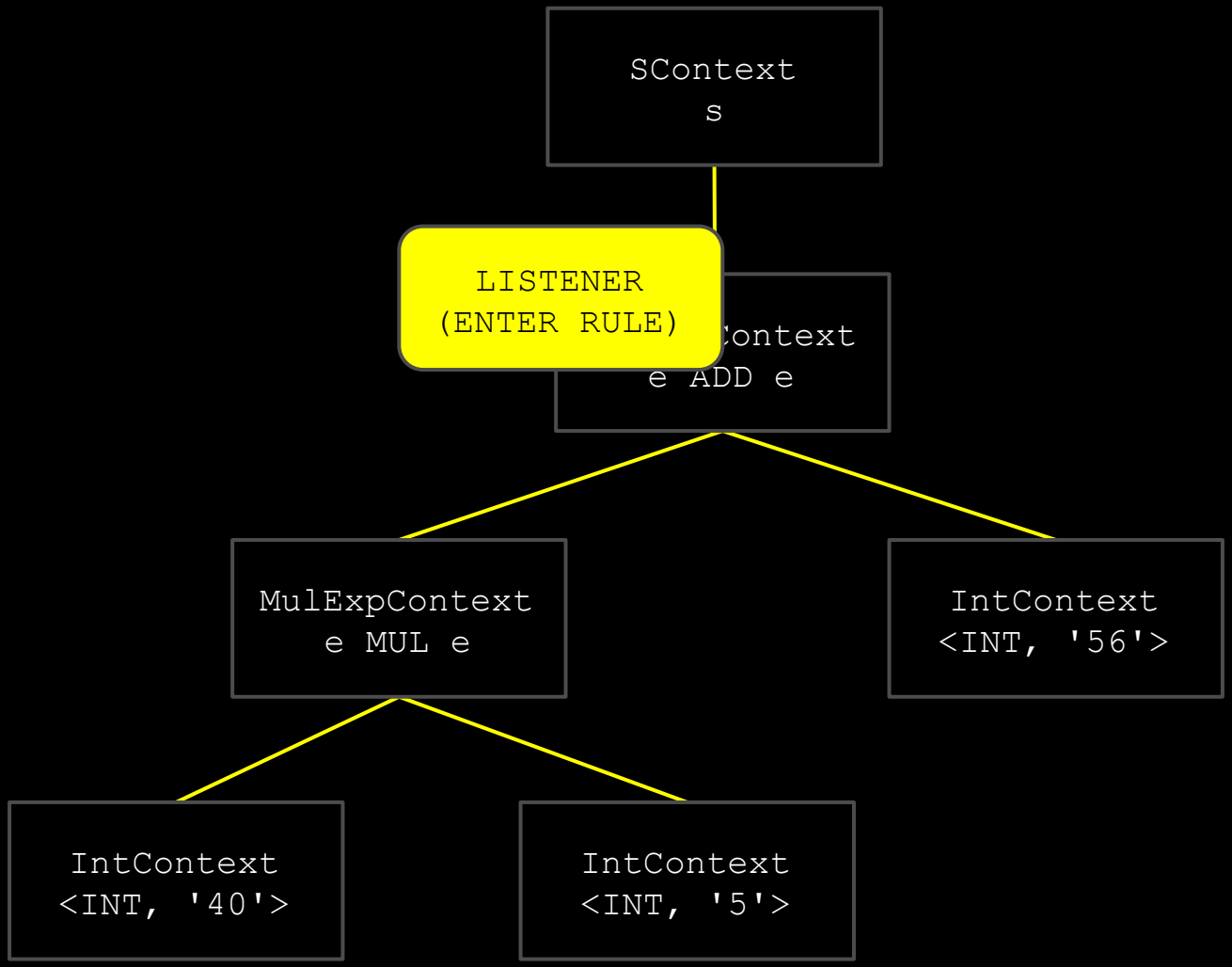
Listener (Exemplo)



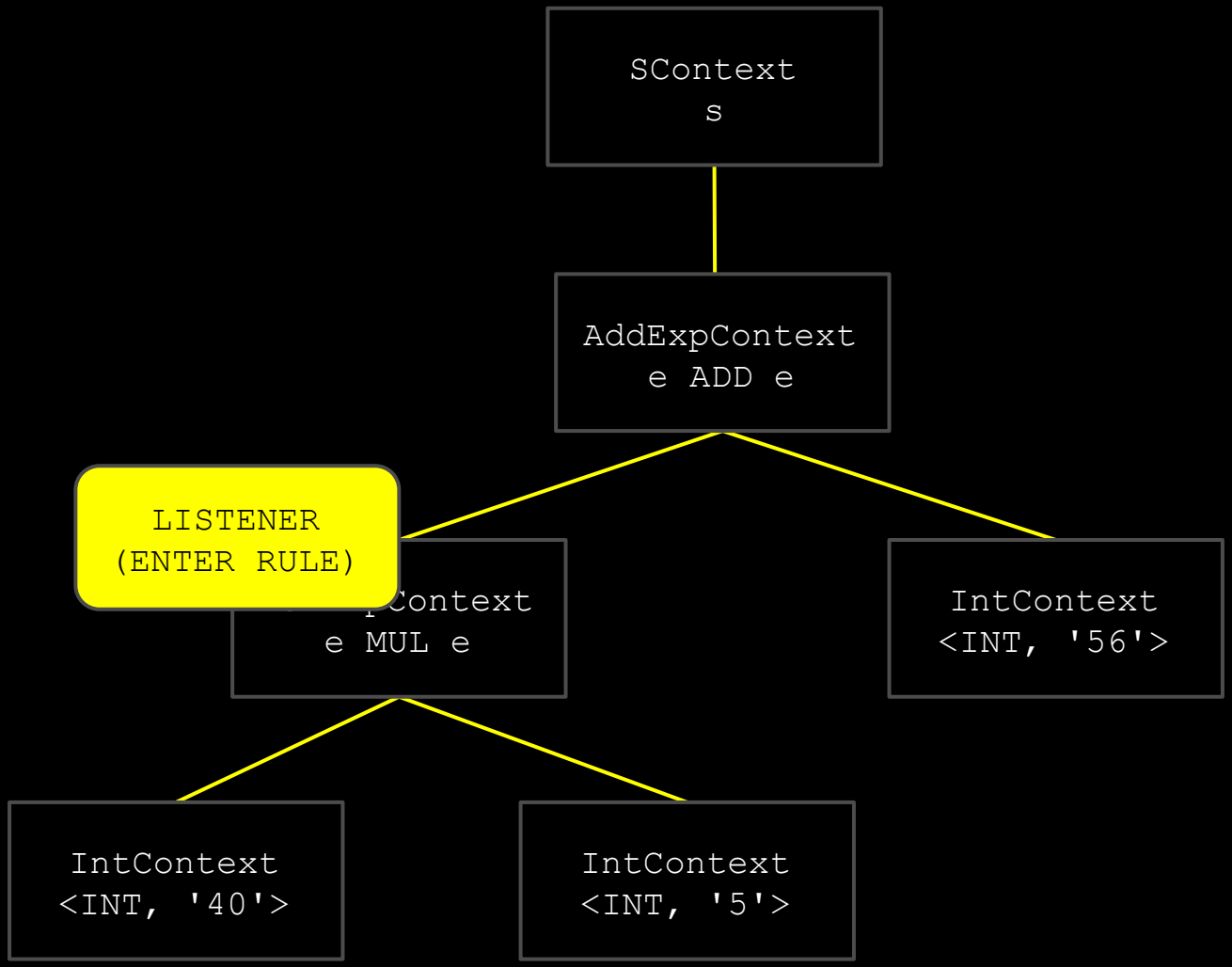
Listener (Exemplo)



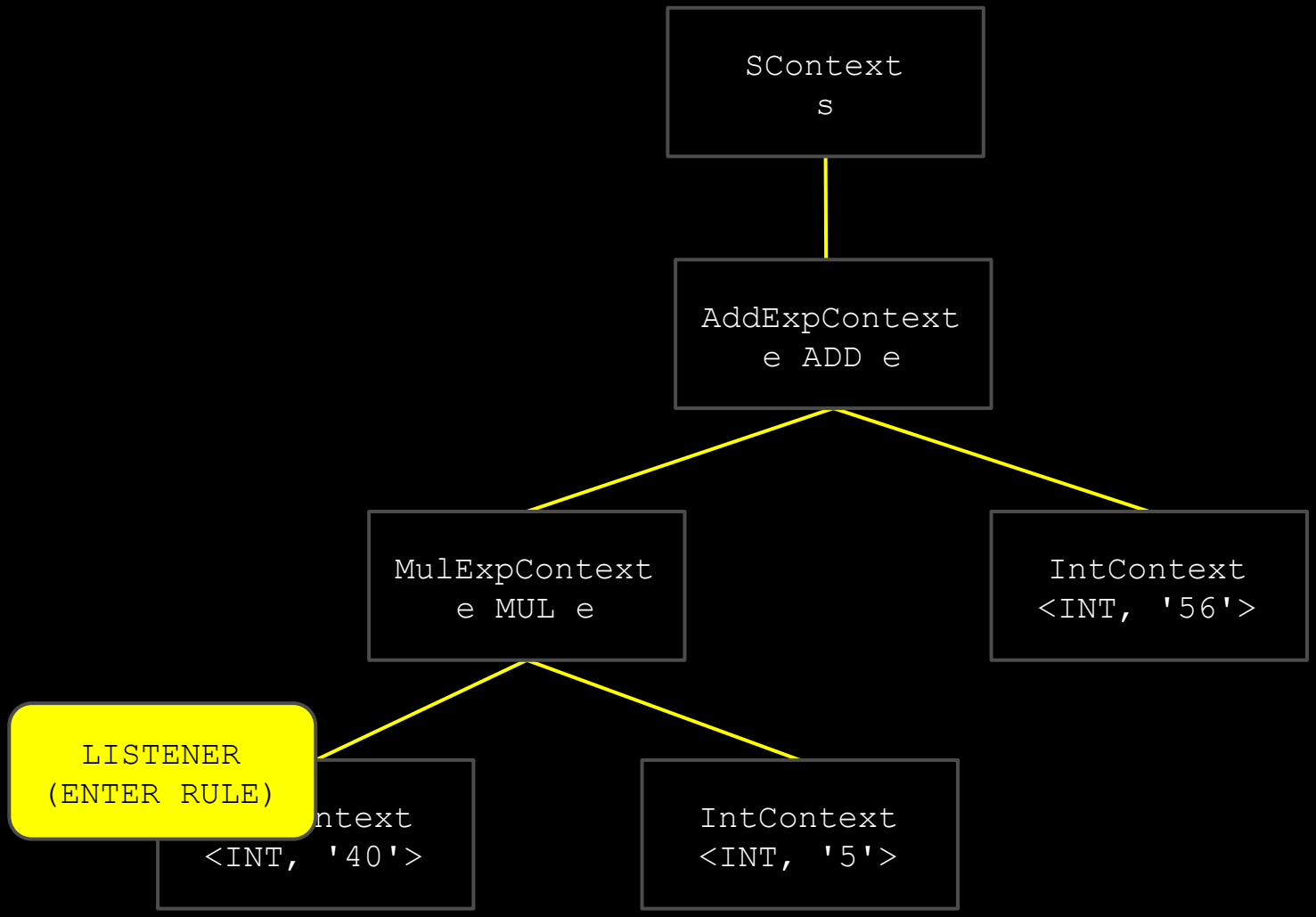
Listener (Exemplo)



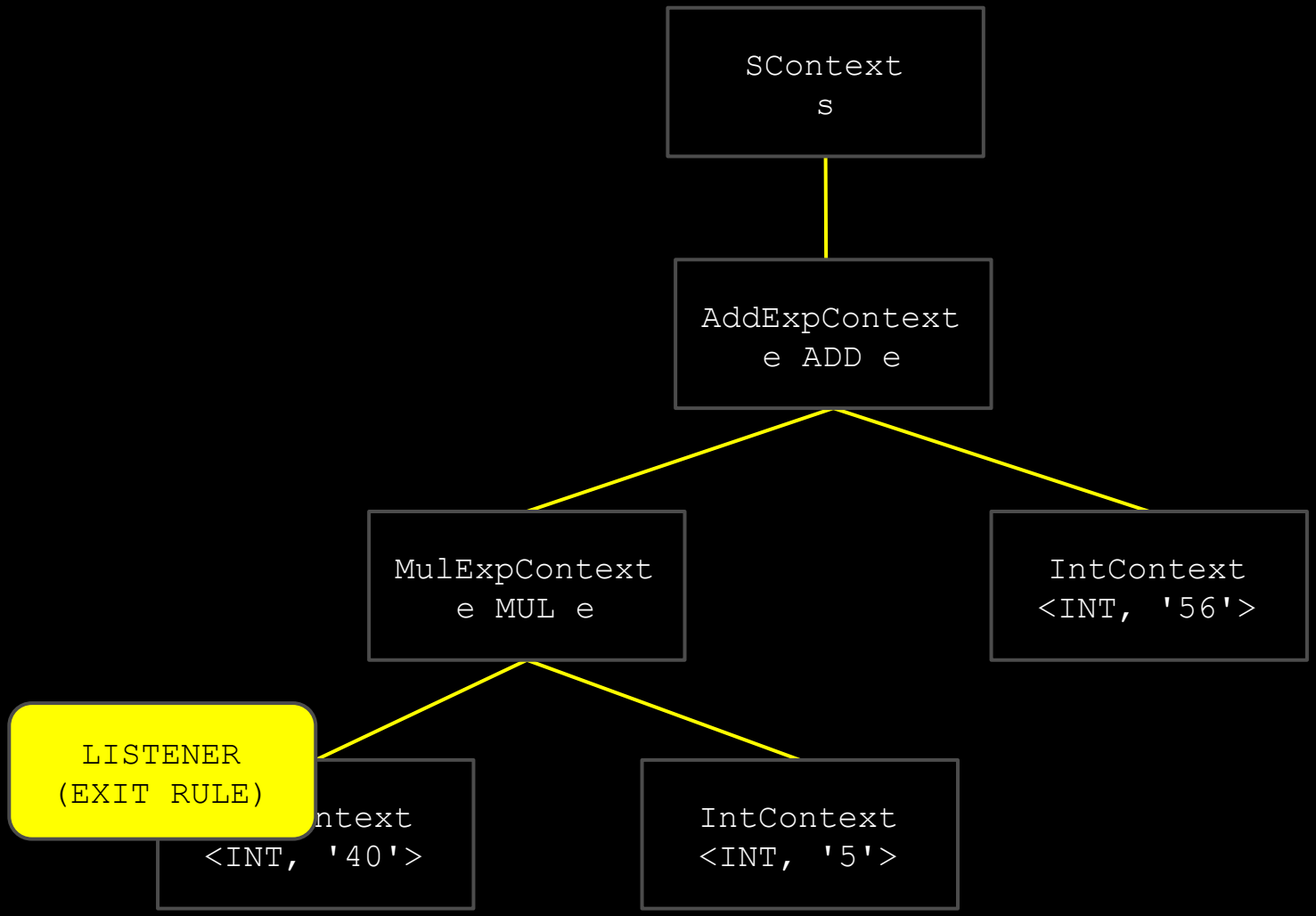
Listener (Exemplo)



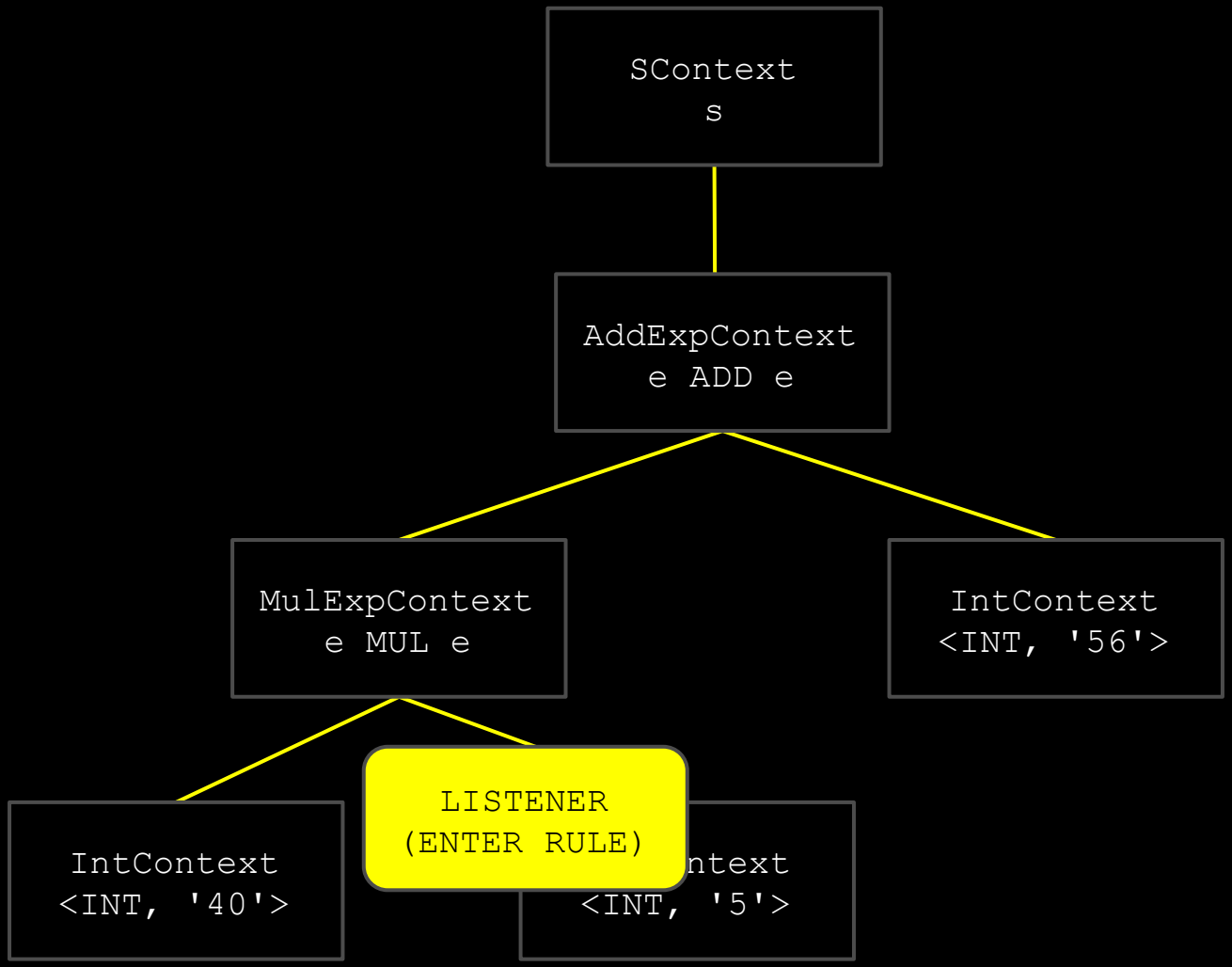
Listener (Exemplo)



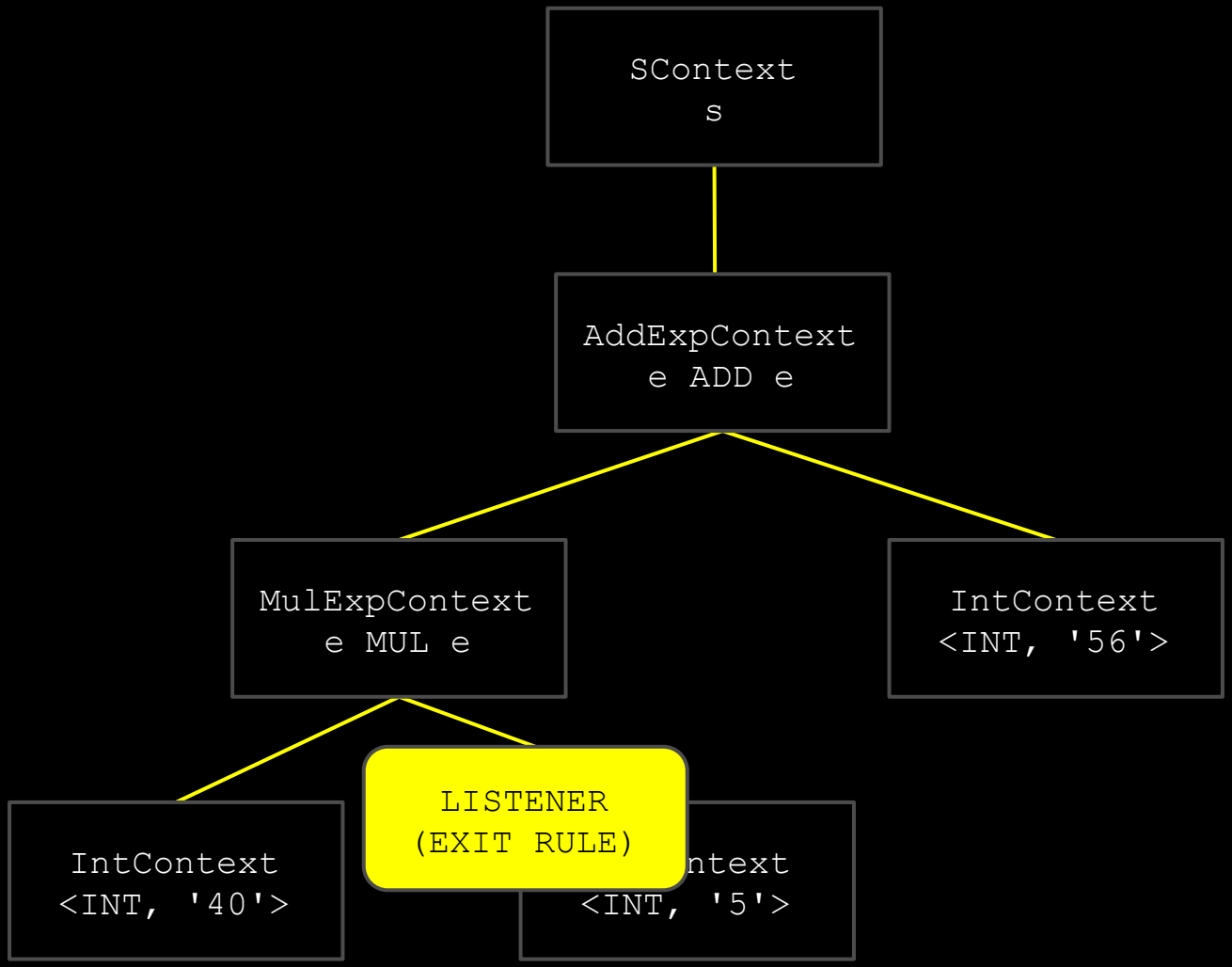
Listener (Exemplo)



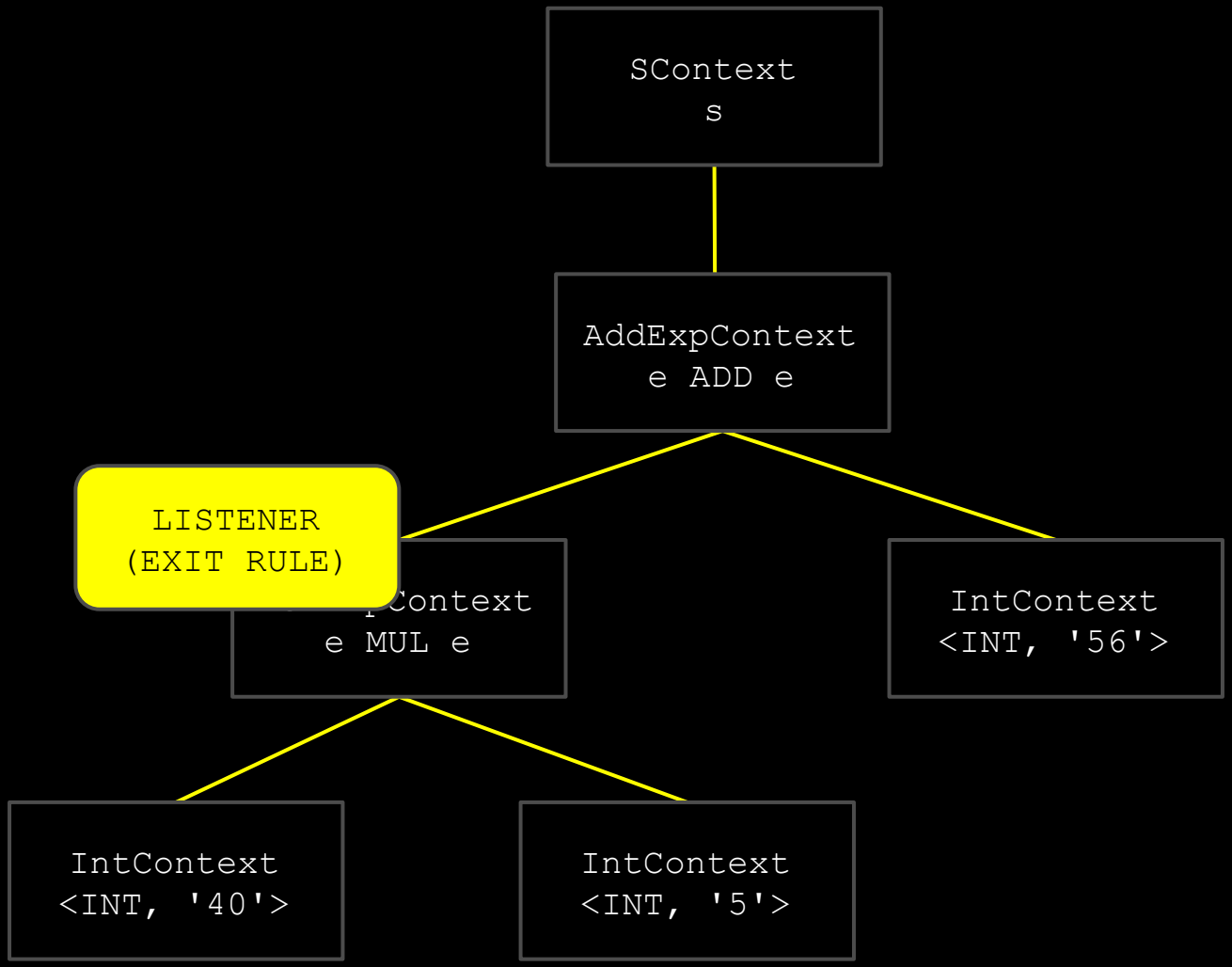
Listener (Exemplo)



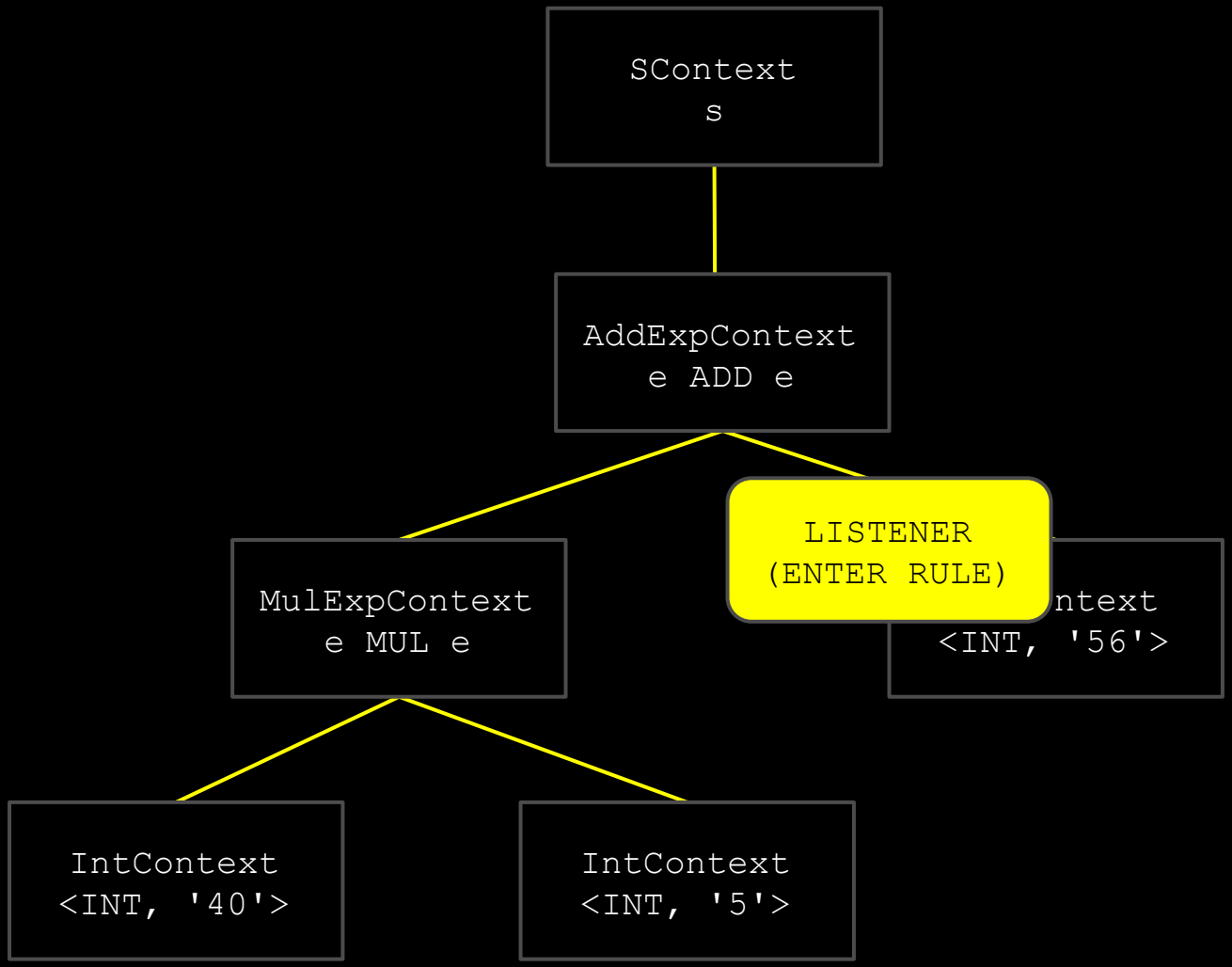
Listener (Exemplo)



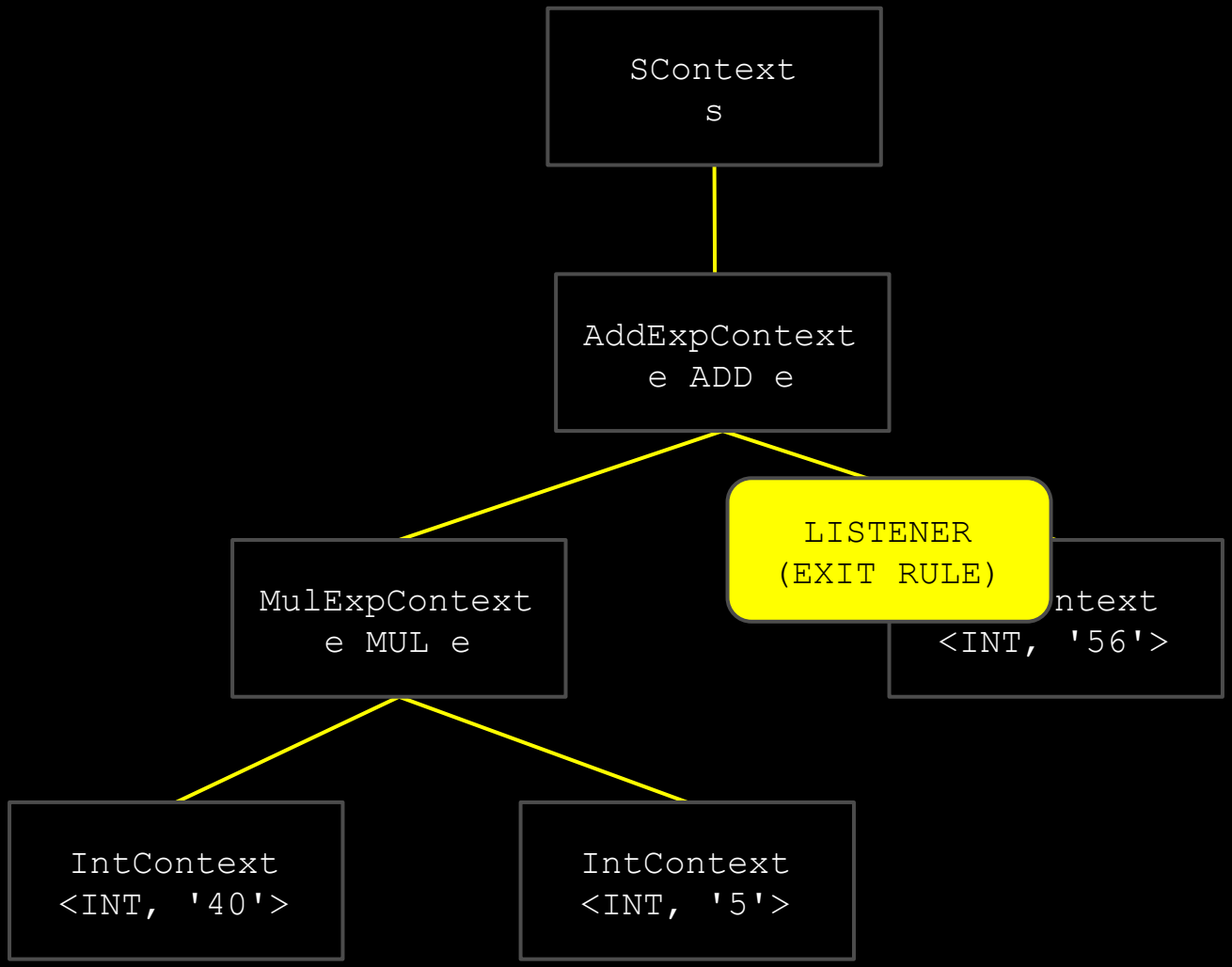
Listener (Exemplo)



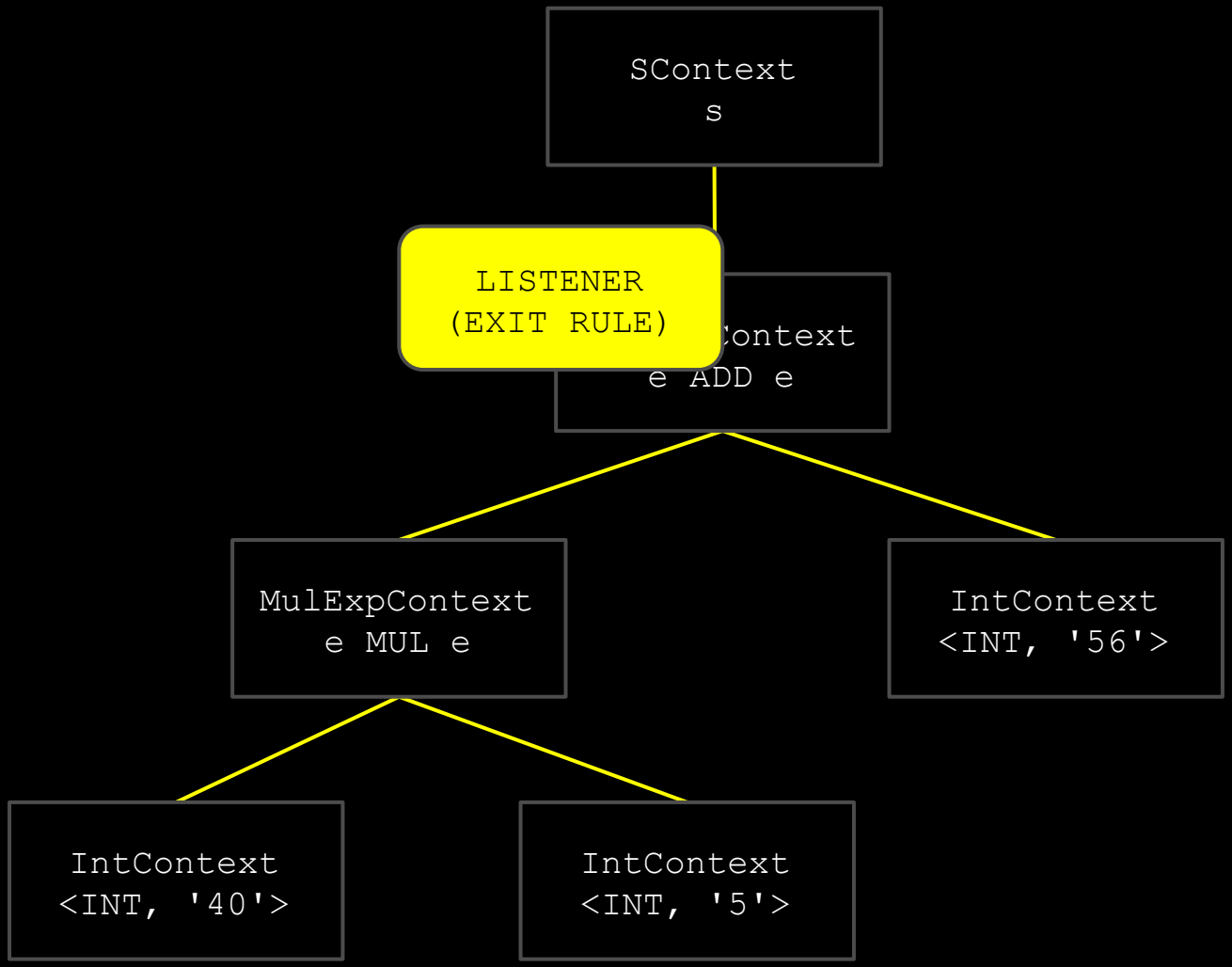
Listener (Exemplo)



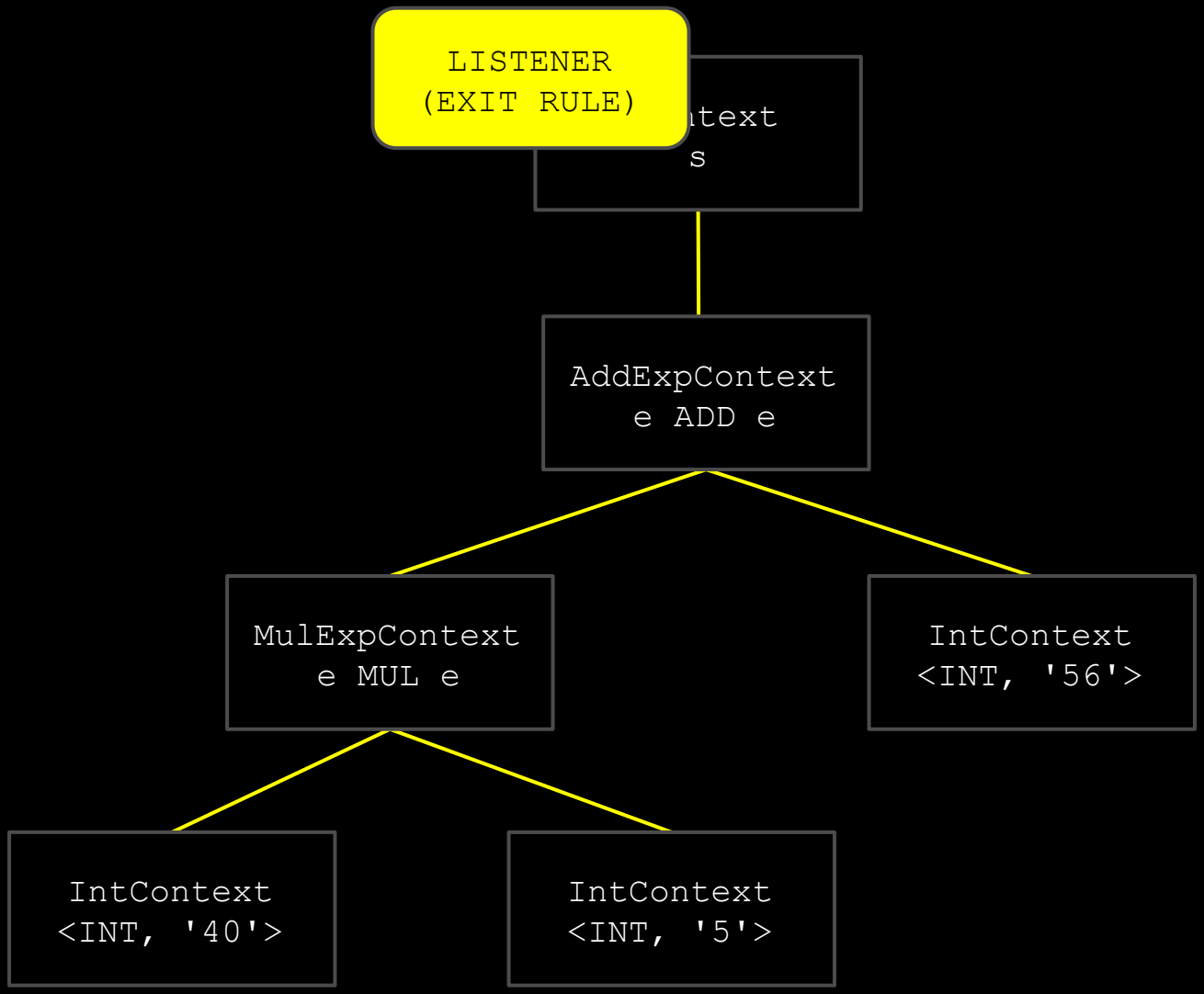
Listener (Exemplo)



Listener (Exemplo)



Listener (Exemplo)



Visitor

- Padrão de projeto comportamental que permite representar uma operação a ser executada nos elementos de uma estrutura de objetos
- Solução que promove a separação do algoritmo da estrutura, encapsulando as operações em um objeto separado (Visitor)

Visitor

- No ANTLR, o visitor pode ser parametrizado para retornar um determinado tipo de classe como resultado da avaliação dos nós visitados
- Normalmente utilizada para avaliações na árvore sintática gerada pelo parser

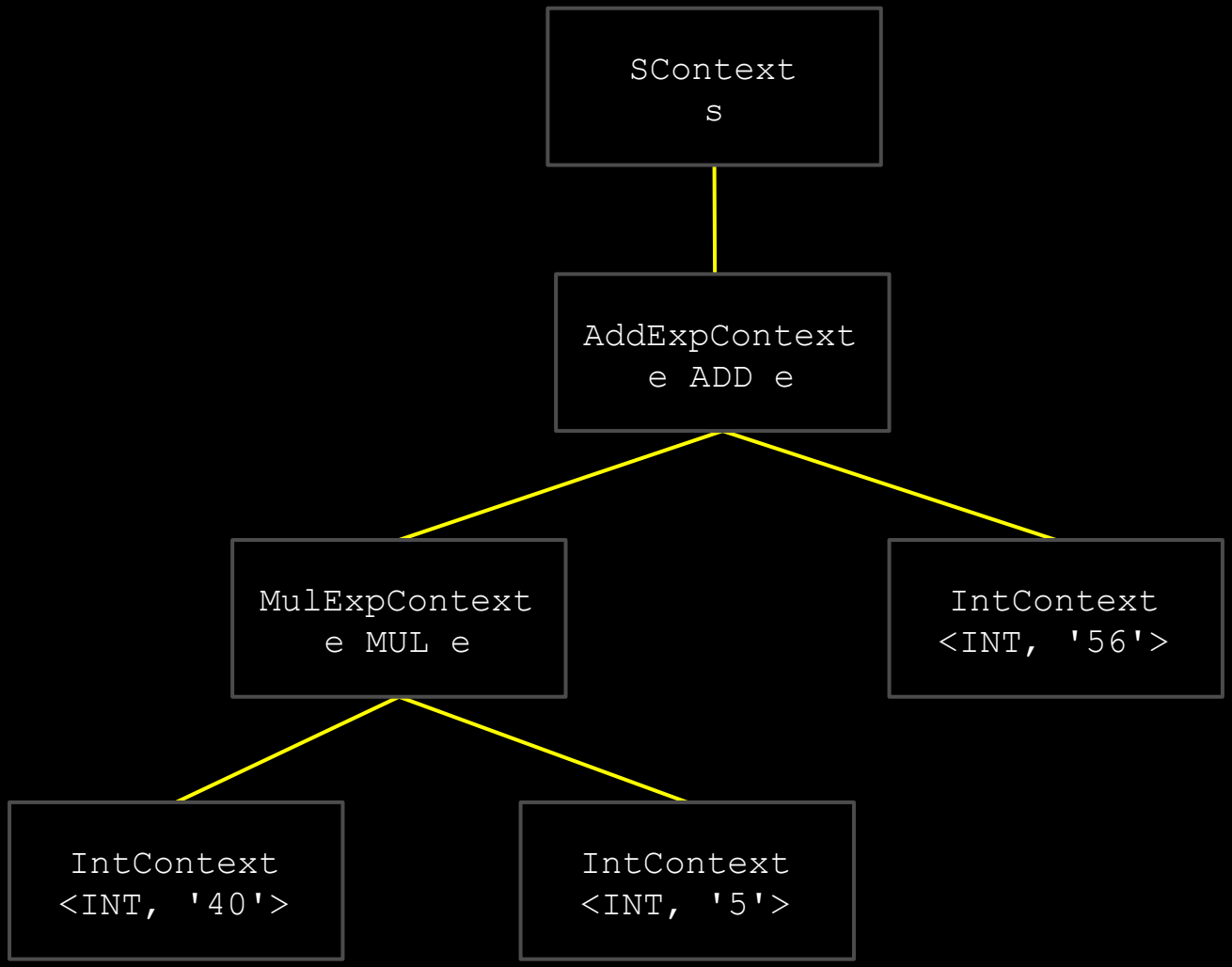
Visitor (Exemplo de Interface de Visitor no ANTLR)

```
public interface AnnotExpVisitor<T>
extends ParseTreeVisitor<T> {
    T visitS(AnnotExpParser.SContext ctx);
    T visitMulExp(AnnotExpParser.MulExpContext ctx);
    T visitAddExp(AnnotExpParser.AddExpContext ctx);
    T visitInt(AnnotExpParser.IntContext ctx);
}
```

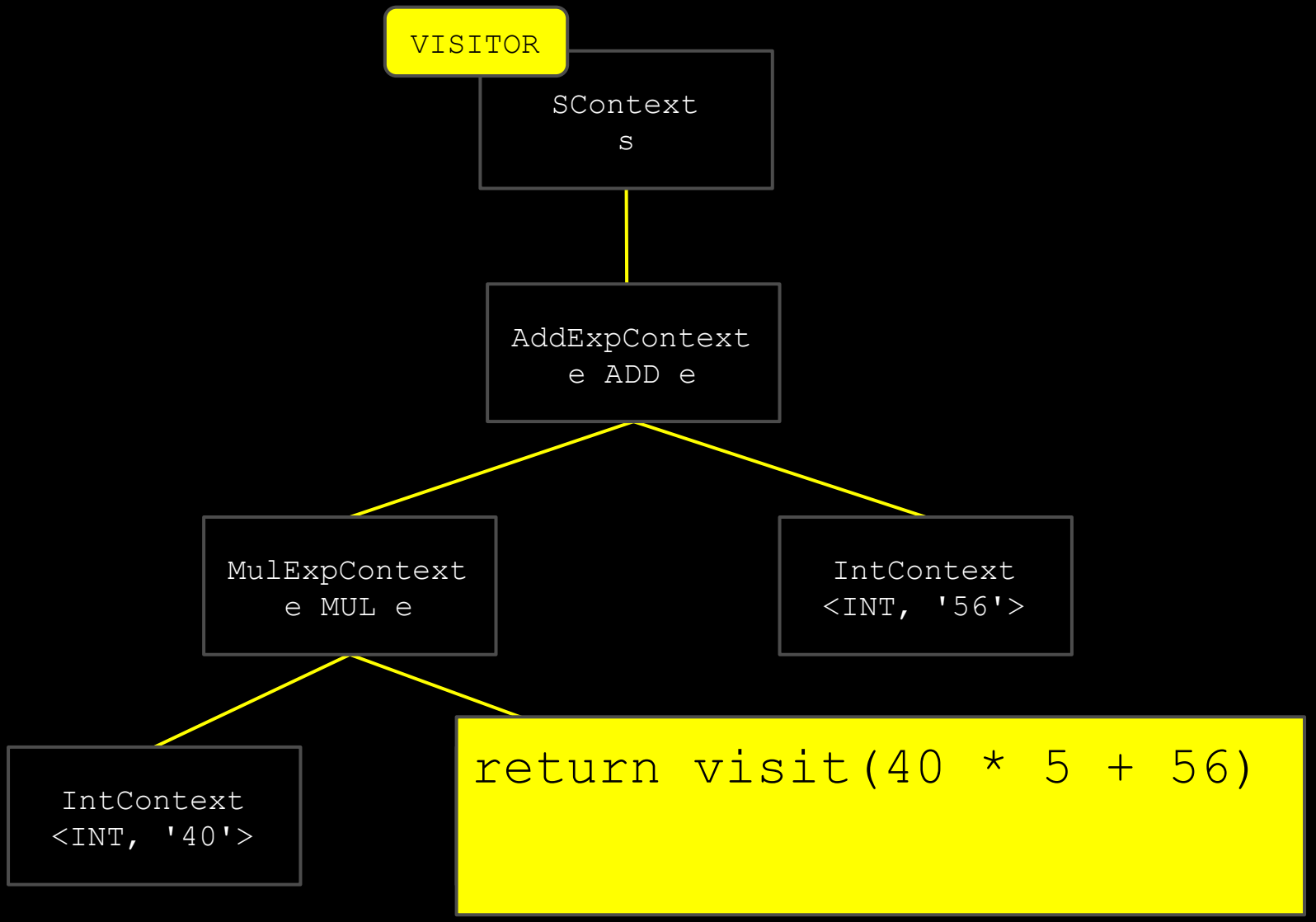
Visitor (Exemplo)

- Percorrendo a árvore sintática gerada pela expressão “40 * 5 + 56” da gramática AnnotExp

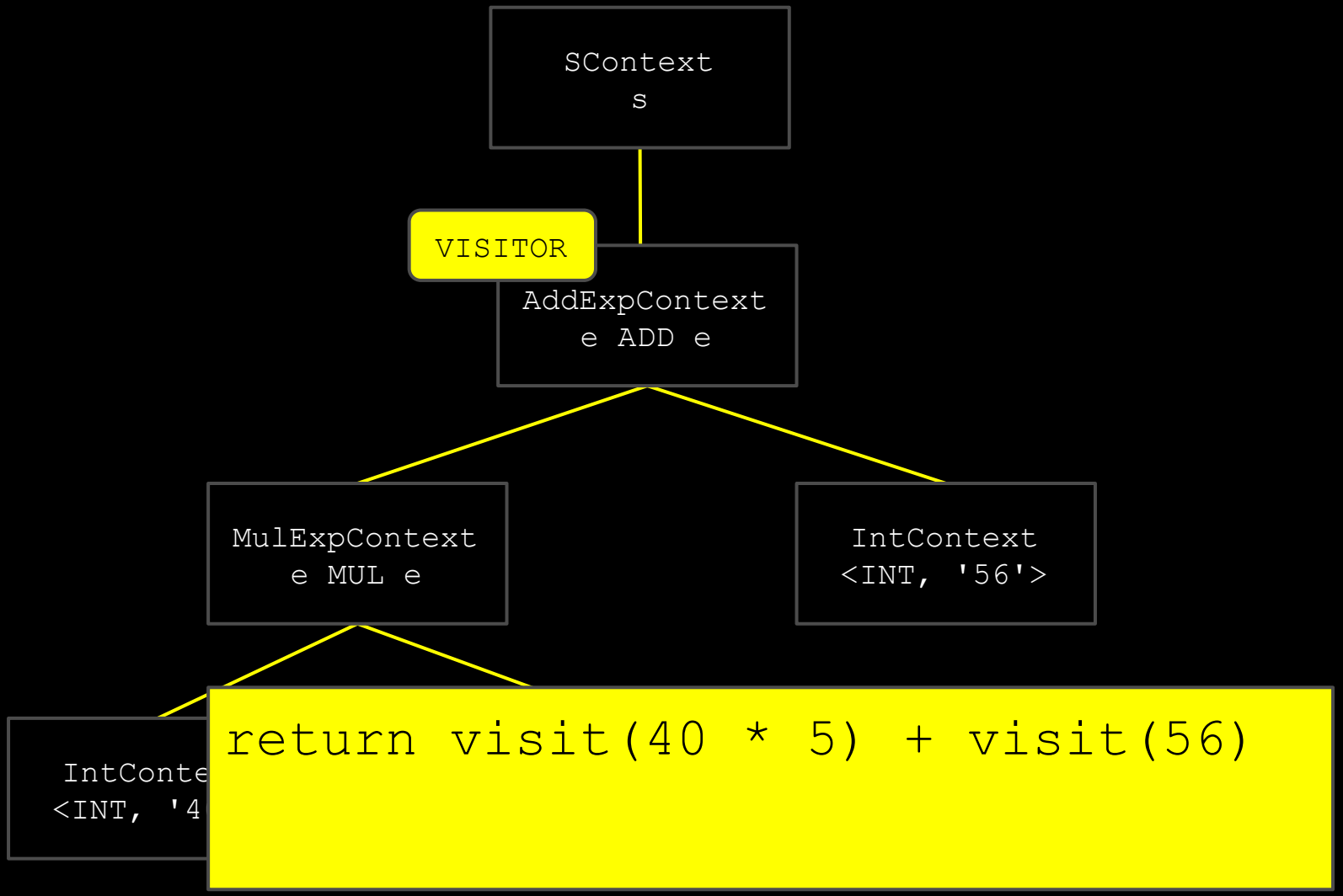
Visitor (Exemplo)



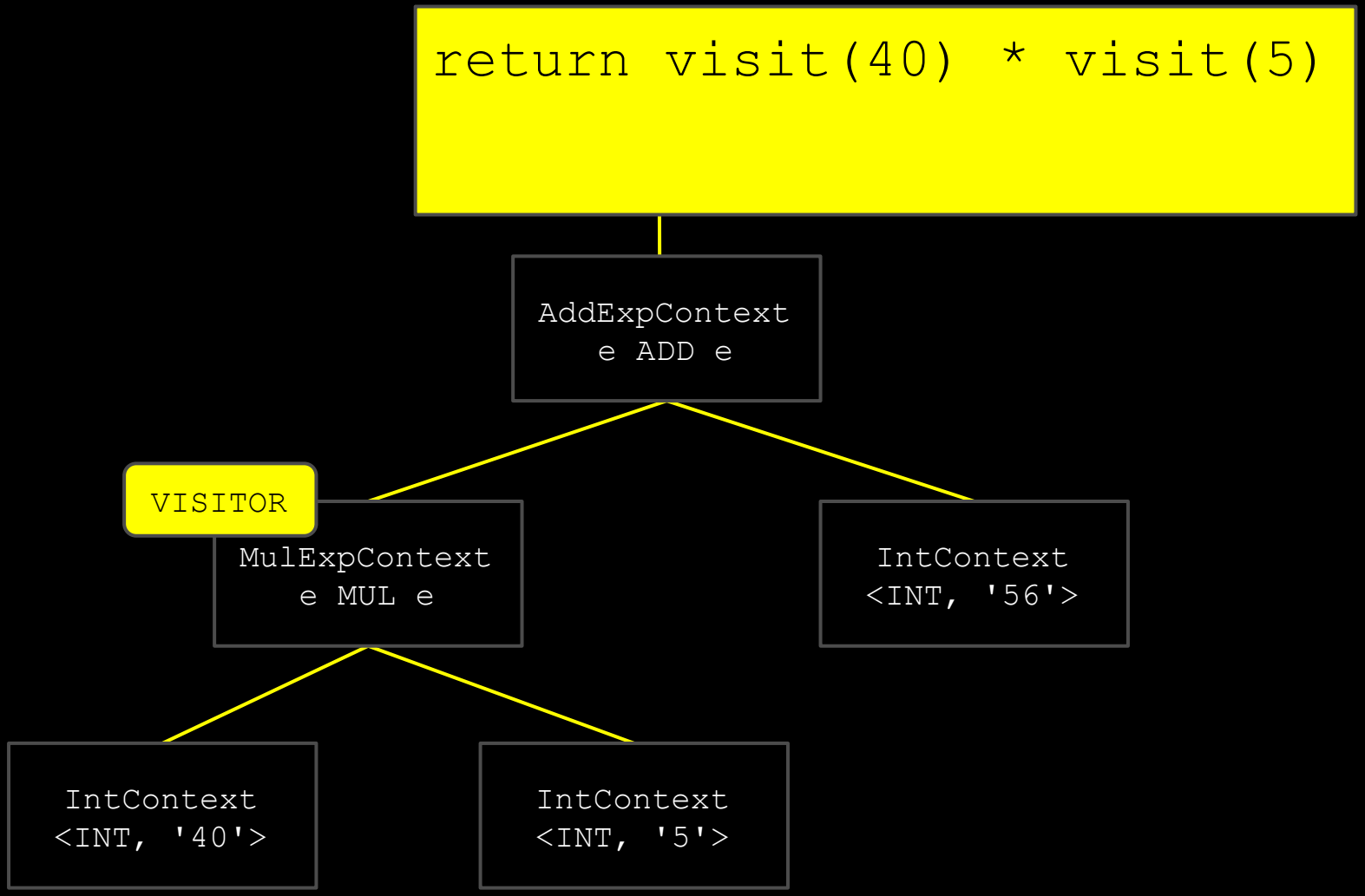
Visitor (Exemplo)



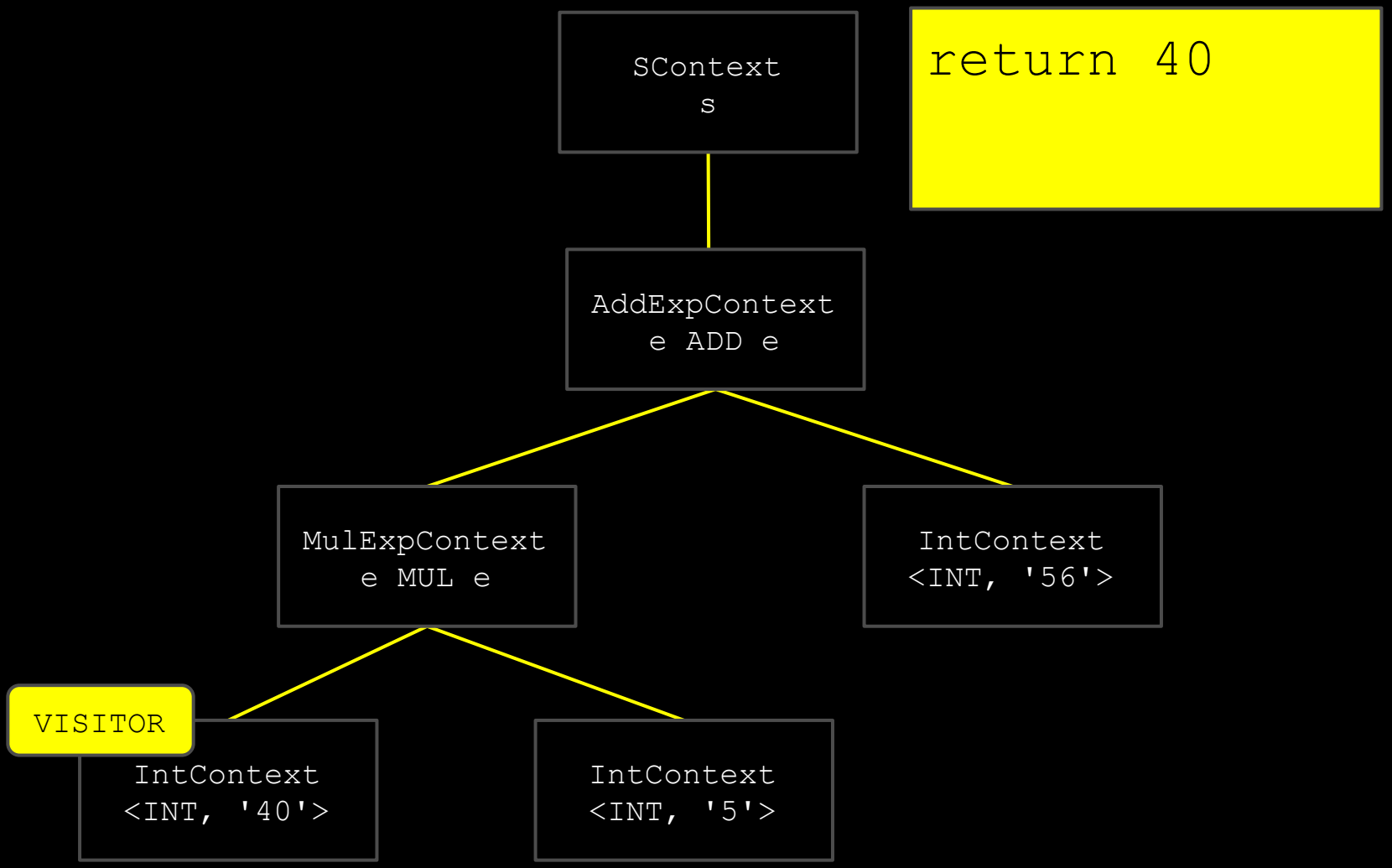
Visitor (Exemplo)



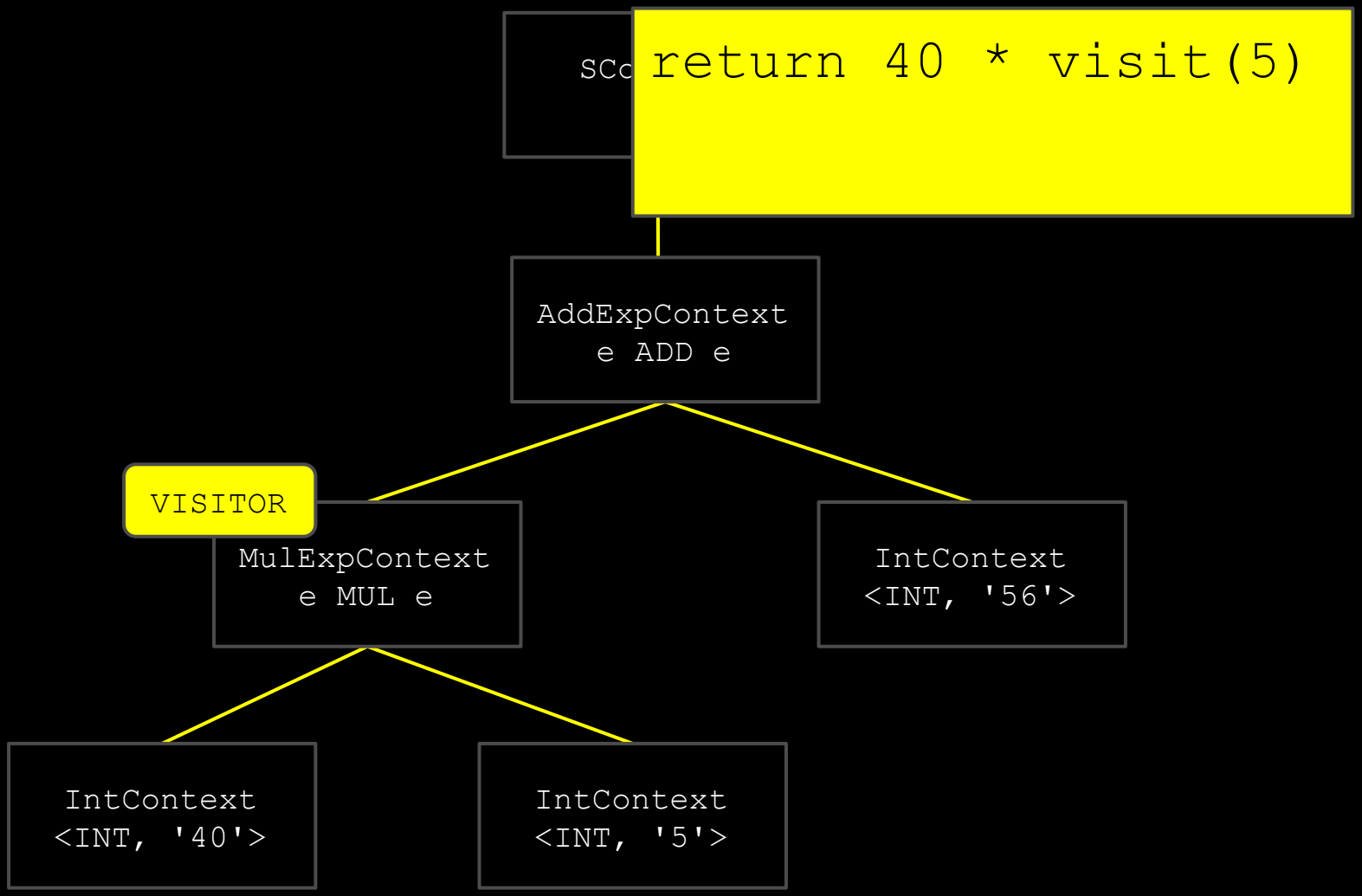
Visitor (Exemplo)



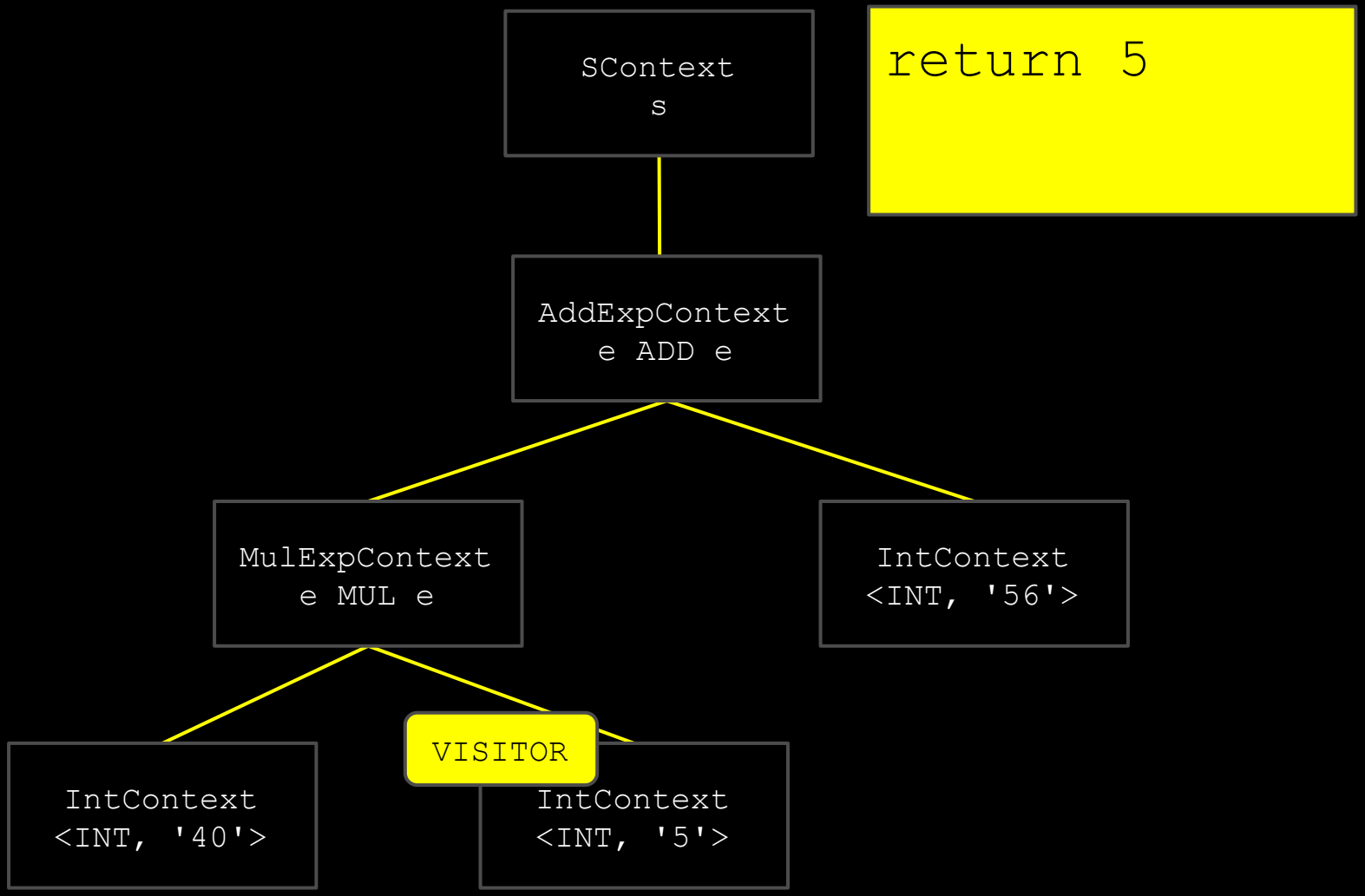
Visitor (Exemplo)



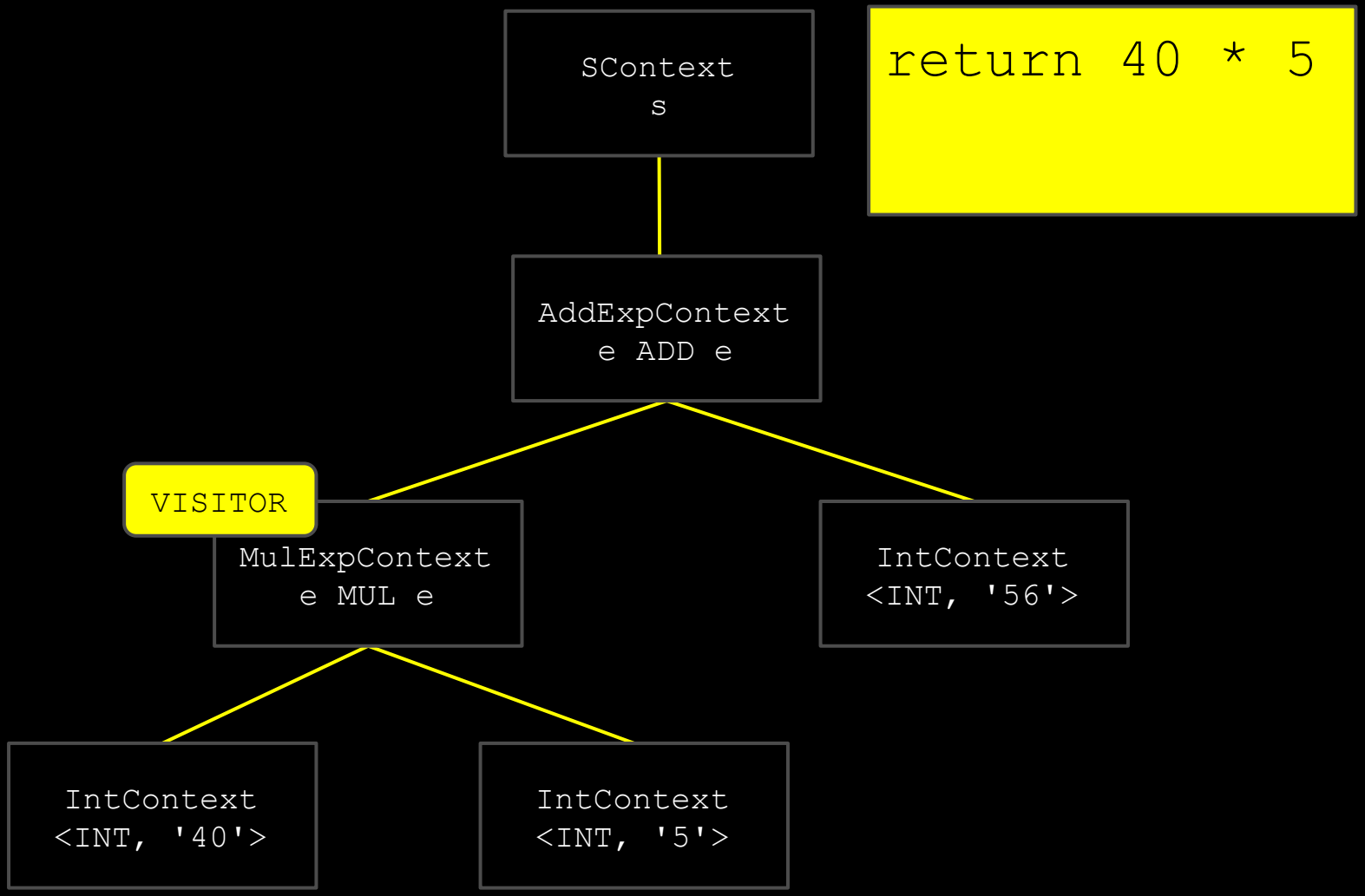
Visitor (Exemplo)



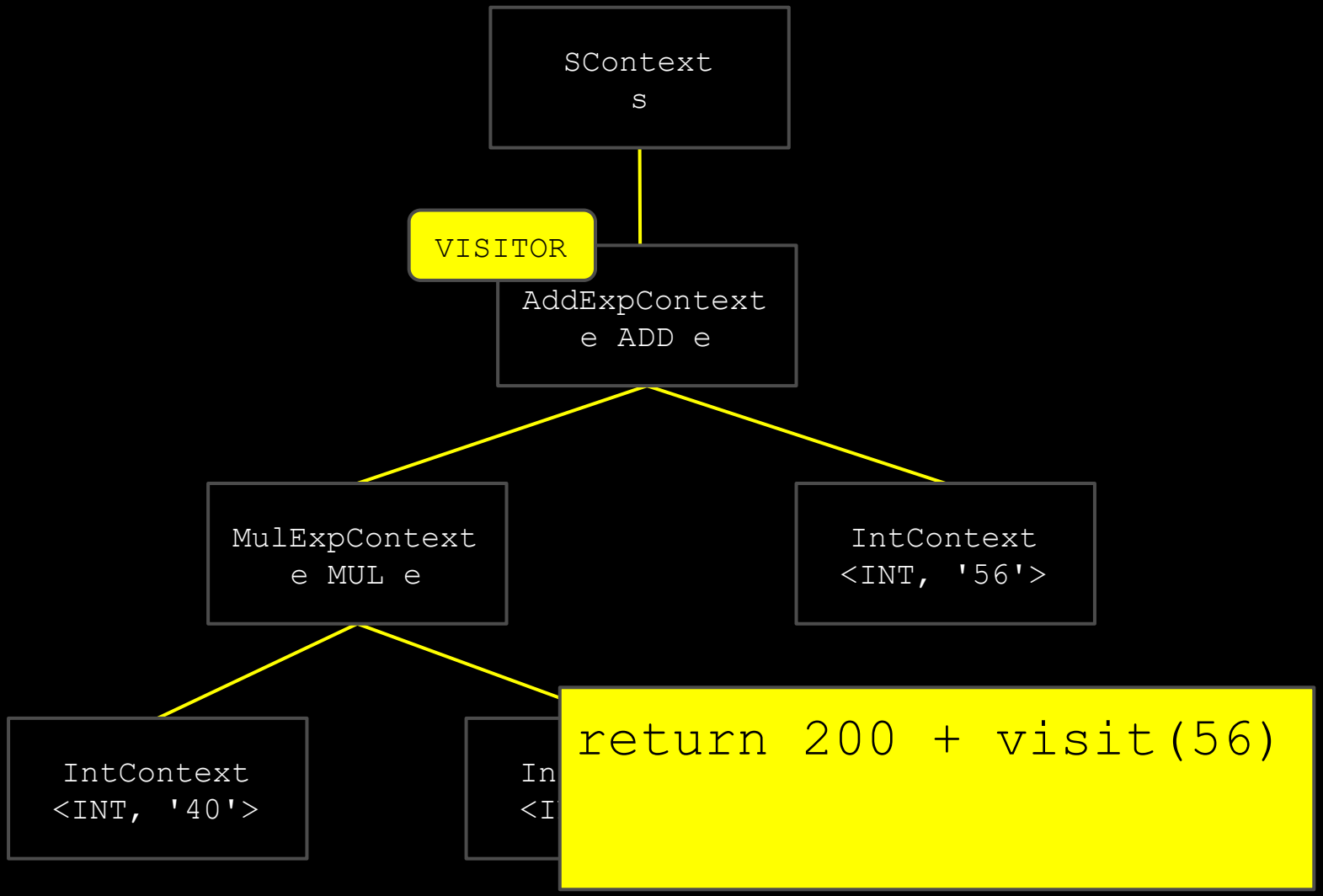
Visitor (Exemplo)



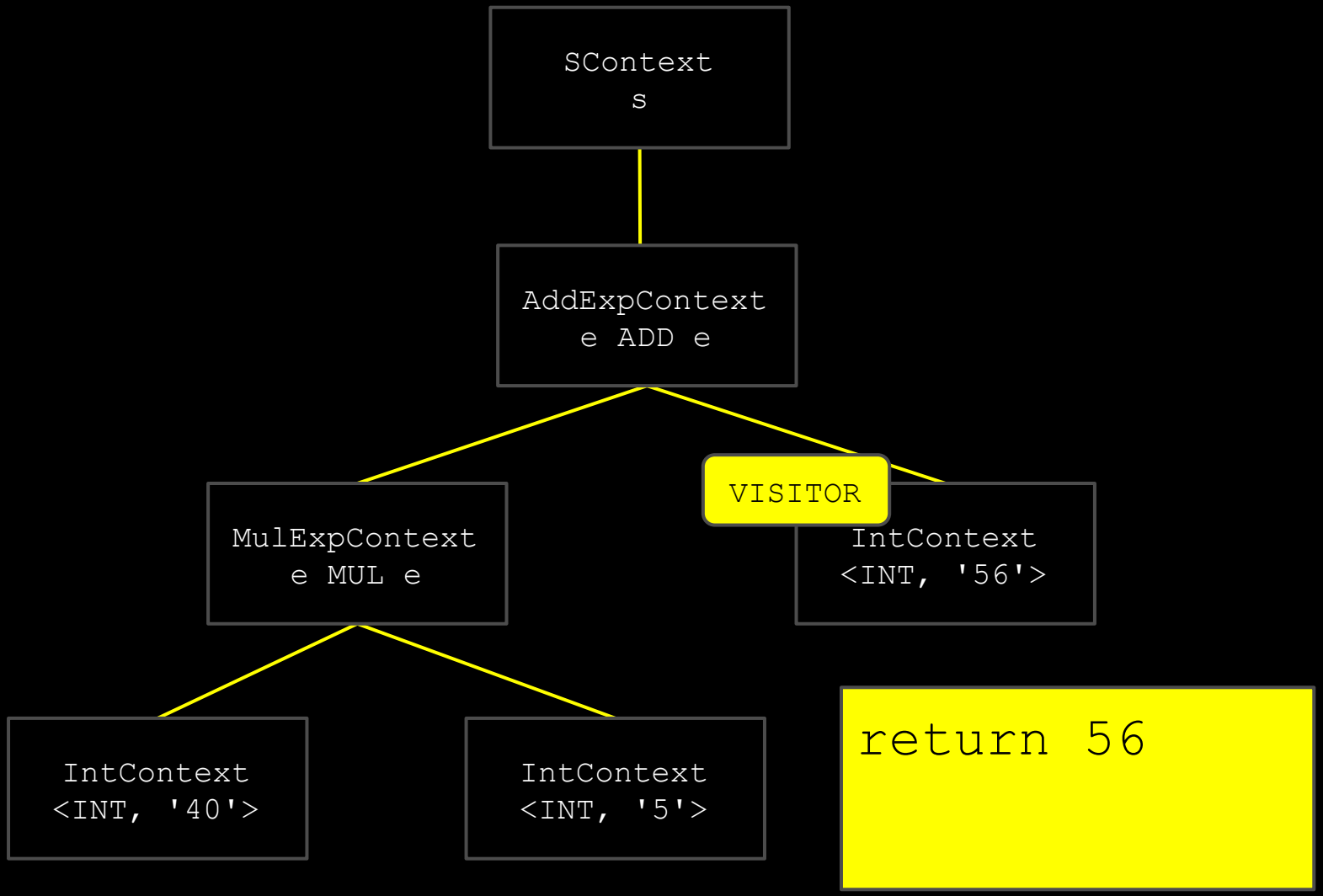
Visitor (Exemplo)



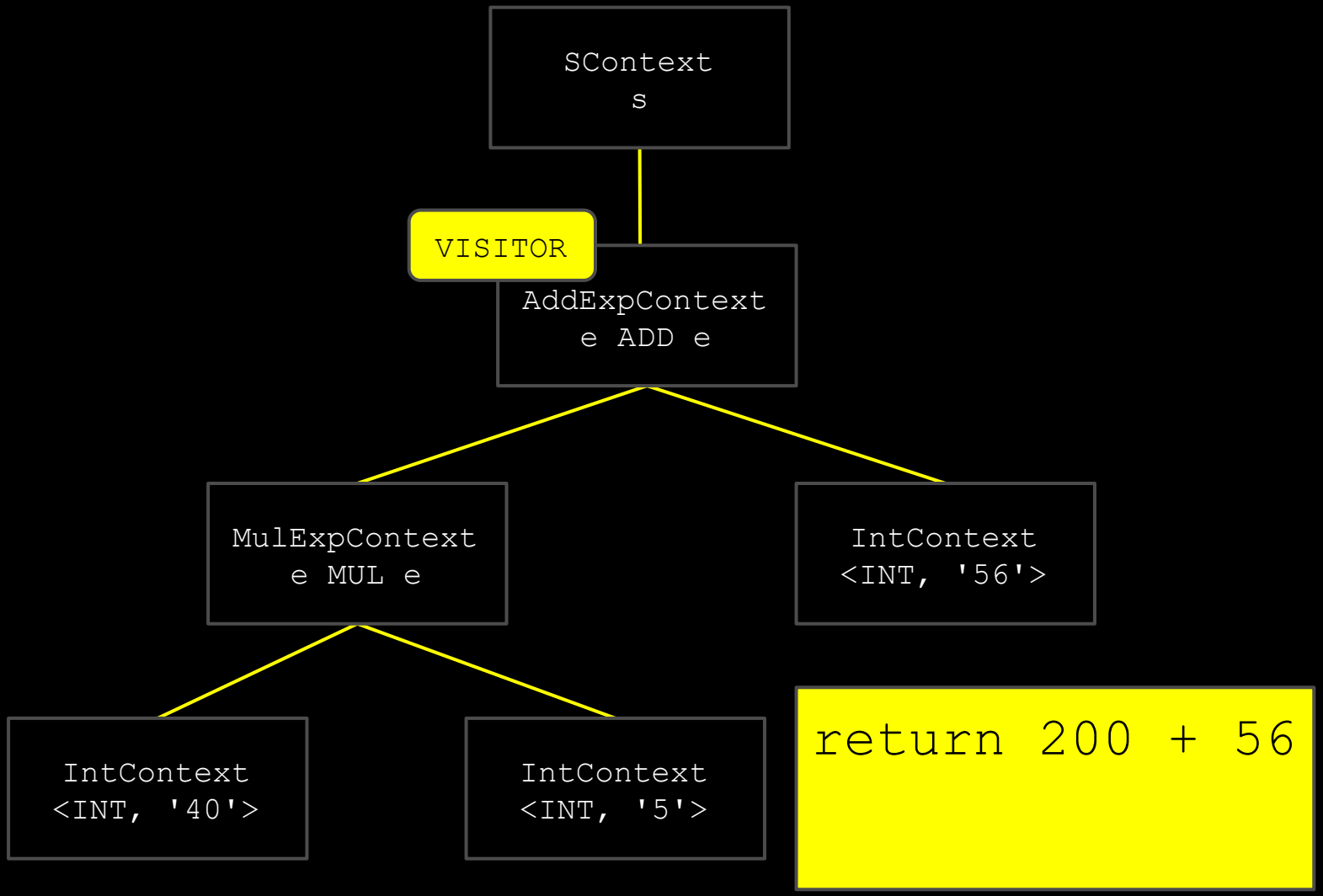
Visitor (Exemplo)



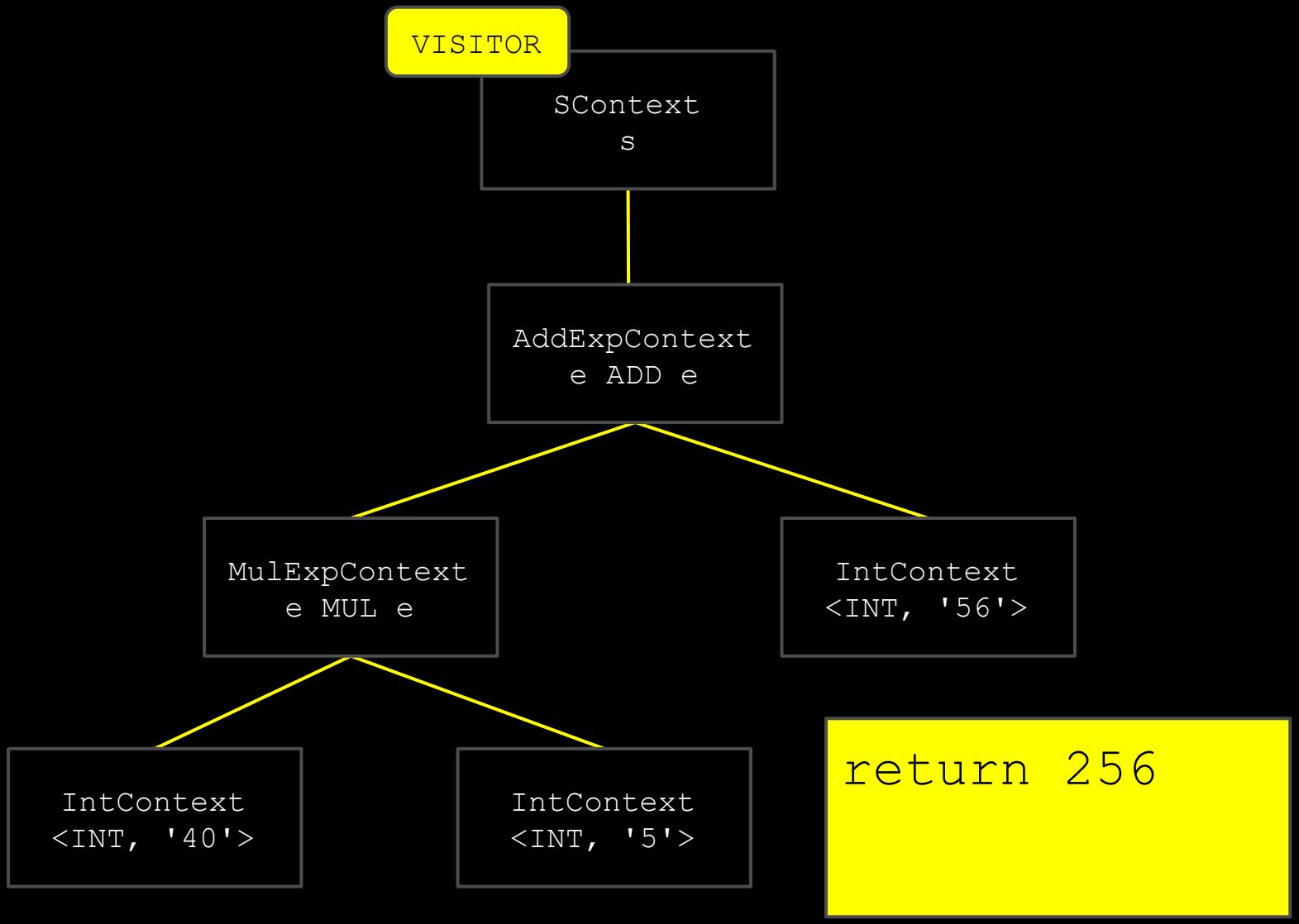
Visitor (Exemplo)



Visitor (Exemplo)



Visitor (Exemplo)



Geração de Arquivos

- Façam o download do ANTLR Versão 4.5.3 a partir do link abaixo: http://www.antlr.org/download/antlr-4.5.3-complete.jar
- Confiram as instruções do arquivo README.txt na pasta compilers-cin\aulas-praticas\ap2\ do repositório da disciplina

Geração de Arquivos (Gramática Test)

```
grammar Test;

stat : ((attrStmt | expr) ';')+ #Expressions
     ;

attrStmt: ID '=' expr           #Assign
        ;
```

Geração de Arquivos (Gramática Test)

```
expr : expr op=('*'|'/') expr    #MultDiv
     | expr op=('+'|'-') expr    #AddSub
     | INT                       #Int
     | ID                        #Identifier
     | '(' expr ')'              #Paren
     ;
//Fragmentos (Não constituem tokens por si só)
fragment NUMBER: [0-9];
fragment LETTER: [a-zA-Z];
fragment UNDERLINE: '_';
```

Geração de Arquivos (Gramática Test)

```
//Tokens
INT : NUMBER+ ;
ID : (UNDERLINE | LETTER) (UNDERLINE |
LETTER | NUMBER)*;

BLOCKCOMMENT : '/*' .*? '*/' -> skip;
LINECOMMENT : '//' .*? '\n' -> skip;
WS :  [ \t\r\n]+ -> skip;
```

Geração de Arquivos

- Arquivos Gerados
  - Test.tokens e TestBase.tokens: Tokens usados no parser
  - TestLexer.java: Analisador léxico (lexer)
  - TestParser.java: Analisador sintático (parser)
  - TestVisitor.java: Interface do visitor
  - TestBaseVisitor.java: Implementação padrão do visitor

Geração de Arquivos (Tokens da Gramática Test)

- O ANTLR cria campos e métodos auxiliares para os tokens
- Deixá-los explícitos pode facilitar bastante

```
expr : expr op=('*'|'/') expr #MultDiv
     | expr op=('+'|'-') expr #AddSub

//Tokens de operações aritméticas
MUL : '*';
DIV : '/';
ADD : '+';
SUB : '-';
```

Geração de Arquivos (Tokens da Gramática Test)

```
T__0=1
T__1=2
T__2=3
T__3=4
INT=5
ID=6
MUL=7
DIV=8
ADD=9
SUB=10
BLOCKCOMMENT=11
LINECOMMENT=12
WS=12
';'=1
'='=2
'('=3
')'=4
'*'=7
'/'=8
'+'=9
'-'=10
```

Geração de Arquivos (Tokens da Gramática Test)

```
T__0=1
T__1=2
T__2=3
T__3=4
INT=5
ID=6
MUL=7
DIV=8
ADD=9
SUB=10
BLOCKCOMMENT=11
```

```
LINECOMMENT=12
WS=12
';'=1
'='=2
'('=3
')'=4
'*'=7
'/'=8
'+'=9
'-'=10
```

Geração de Arquivos (Tokens da Gramática Test)

```
T__0=1
T__1=2
T__2=3
T__3=4
INT=5
ID=6
MUL=7
DIV=8
ADD=9
SUB=10
BLOCKCOMMENT=11
LINECOMMENT=12
WS=12
';'=1
'='=2
'('=3
')'=4
'*'=7
'/'=8
'+'=9
'-'=10
```

Exercícios Práticos

1. Modifique o visitor TestExtendVisitor para computar o resultado das expressões definidas na gramática Test.
Ex.:

```
x = (3 - 4) * 5; //Entrada
x = -5           //Saída


x = 3 * 4 + 5;    //Entrada (linha 1)
y = x + 10;       //Entrada (linha 2)
x = 17            //Saída (linha 1)
y = 27            //Saída (linha 2)
```

Exercícios Práticos

Para cada variável não declarada, o visitor deverá exibir uma mensagem de erro indicando o nome, a linha e a coluna na qual a variável foi encontrada na entrada. Caso não exista nenhuma variável não declarada, então, ao final da avaliação de todas as expressões de entrada, o visitor deverá exibir os valores finais de todas as variáveis declaradas.

Exercícios Práticos

```
x = 10;                 //Entrada (linha 1)
x = x * 20;             //Entrada (linha 2)
x = 200                 //Saída


x = (3 - 4) * 5; //Entrada (linha 1)
y = z + 10;             //Entrada (linha 2)
x = 10 + w;             //Entrada (linha 3)
erro: variável 'z' indefinida na linha
2, coluna 4
erro: variável 'w' indefinida na linha
3, coluna 9
```

Exercícios Práticos

- O exercício prático deve ser realizado individualmente e enviado por e-mail com o assunto “EXERCÍCIOS PRÁTICOS 02” para monitoria-if688-l@cin.ufpe.br até as 23:59 de amanhã (20.09.2017)
- A resolução do exercício prático deve estar em um arquivo com o nome “TestExtendVisitor.java”