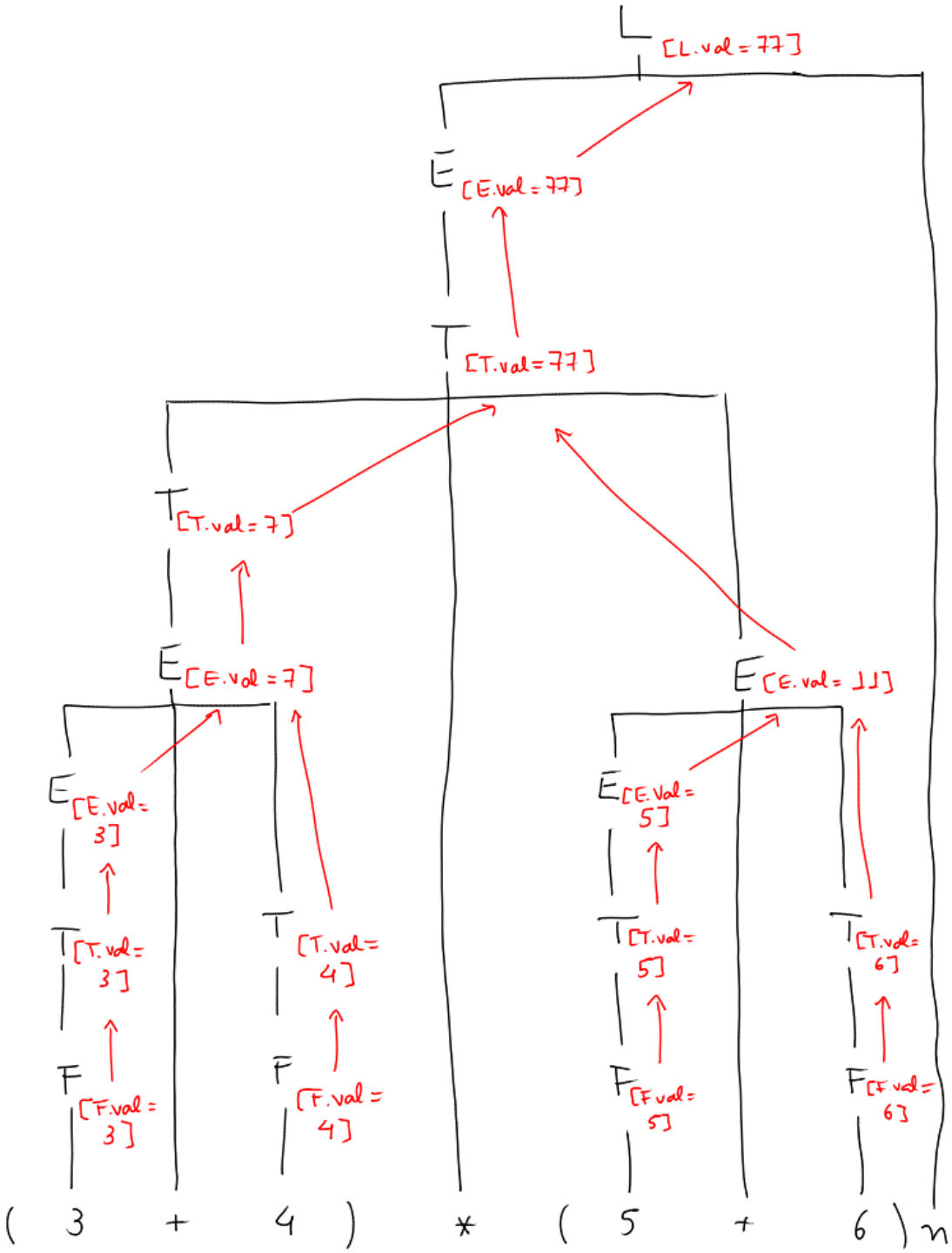
Construção de Compiladores 1 - 2015.1 - Prof. Daniel Lucrédio
Lista 07 - Análise semântica

1) Considere a DDS abaixo

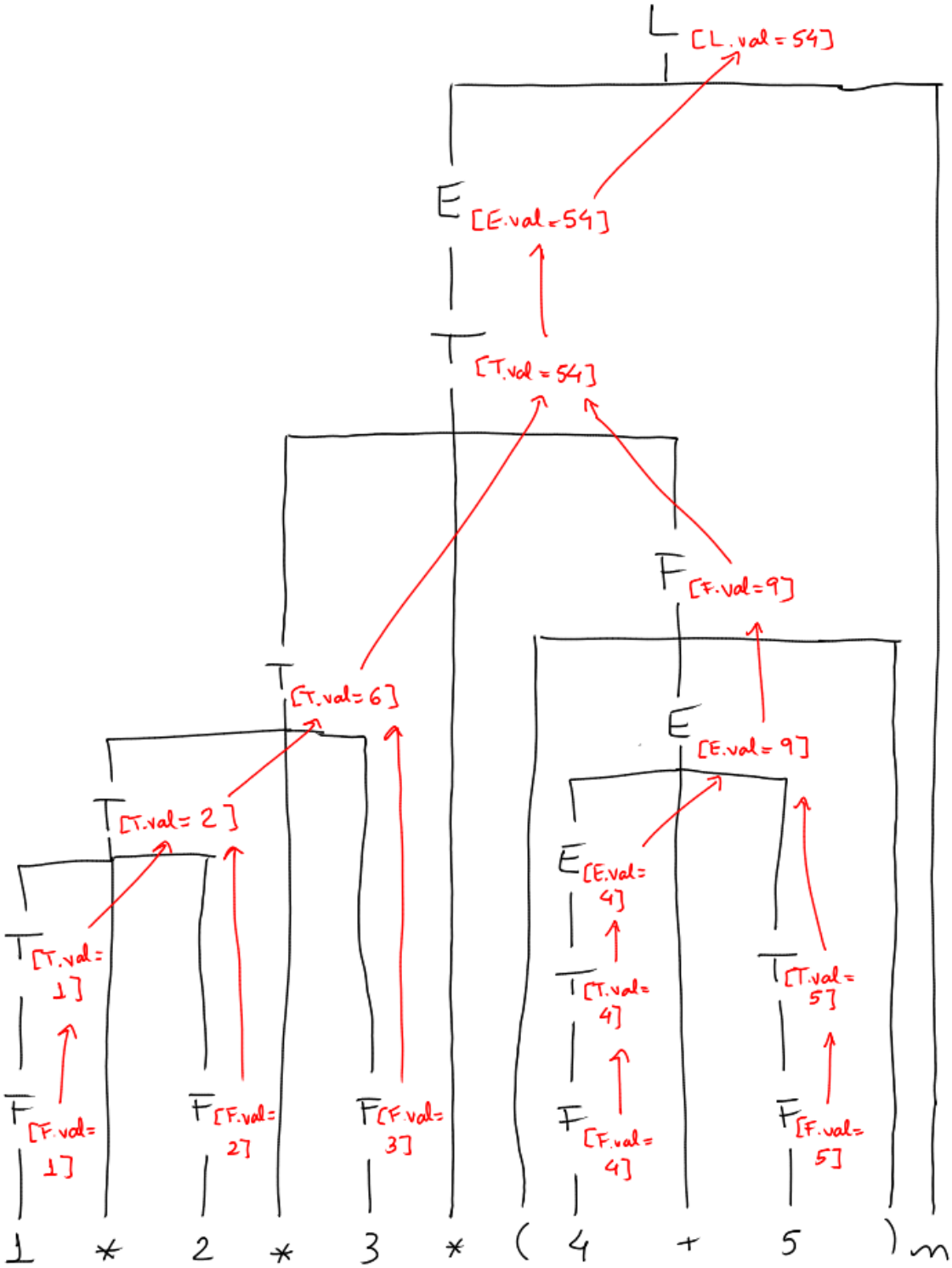
Produção	Regras semânticas
`L → E n`	`L.val = E.val`
`E → E1 + T`	`E.val = E1.val + T.val`
`E → T`	`E.val = T.val`
`T → T1 * F`	`T.val = T1.val * F.val`
`T → F`	`T.val = F.val`
`F → ( E )`	`F.val = E.val`
`F → dígito`	`F.val = dígito.lexval`

Construa as árvores sintáticas com o cálculo dos atributos para as seguintes expressões:

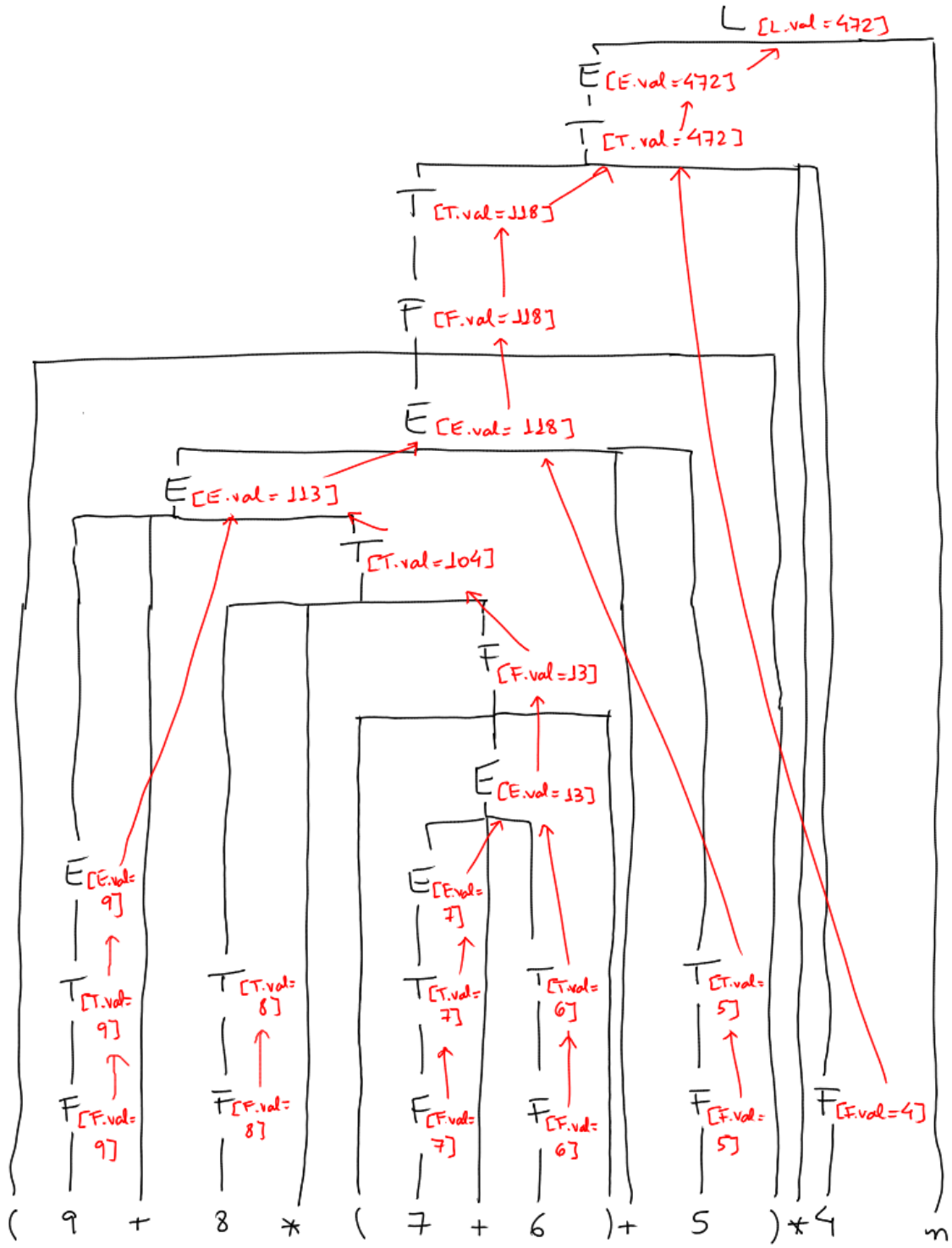
a) (3 + 4) * (5 + 6) n


L [L.val = 77]
E [E.val = 77]
T [T.val = 77]
T [T.val = 7]
E [E.val = 7]
E [E.val = 11]
E [E.val = 3]
T [T.val = 3]
F [F.val = 3]
T [T.val = 4]
F [F.val = 4]
E [E.val = 5]
T [T.val = 5]
F [F.val = 5]
T [T.val = 6]
F [F.val = 6]
(3 + 4) * (5 + 6) n

b) 1 * 2 * 3 * (4 + 5) n


L [L.val = 54]
E [E.val = 54]
T [T.val = 54]
T [T.val = 6]
T [T.val = 2]
T [T.val = 1]
F [F.val = 1]
F [F.val = 2]
F [F.val = 3]
F [F.val = 9]
E [E.val = 9]
E [E.val = 4]
T [T.val = 4]
F [F.val = 4]
T [T.val = 5]
F [F.val = 5]
1 * 2 * 3 * (4 + 5) n

c) (9 + 8 * (7 + 6) + 5) * 4n


L [L.val = 472]
E [E.val = 472]
T [T.val = 472]
T [T.val = 118]
F [F.val = 118]
E [E.val = 118]
E [E.val = 113]
T [T.val = 104]
F [F.val = 13]
E [E.val = 13]
E [E.val = 9]
T [T.val = 9]
F [F.val = 9]
T [T.val = 8]
F [F.val = 8]
E [E.val = 7]
T [T.val = 7]
F [F.val = 7]
T [T.val = 6]
F [F.val = 6]
T [T.val = 5]
F [F.val = 5]
F [F.val = 4]
(9 + 8 * (7 + 6) + 5) * 4 n

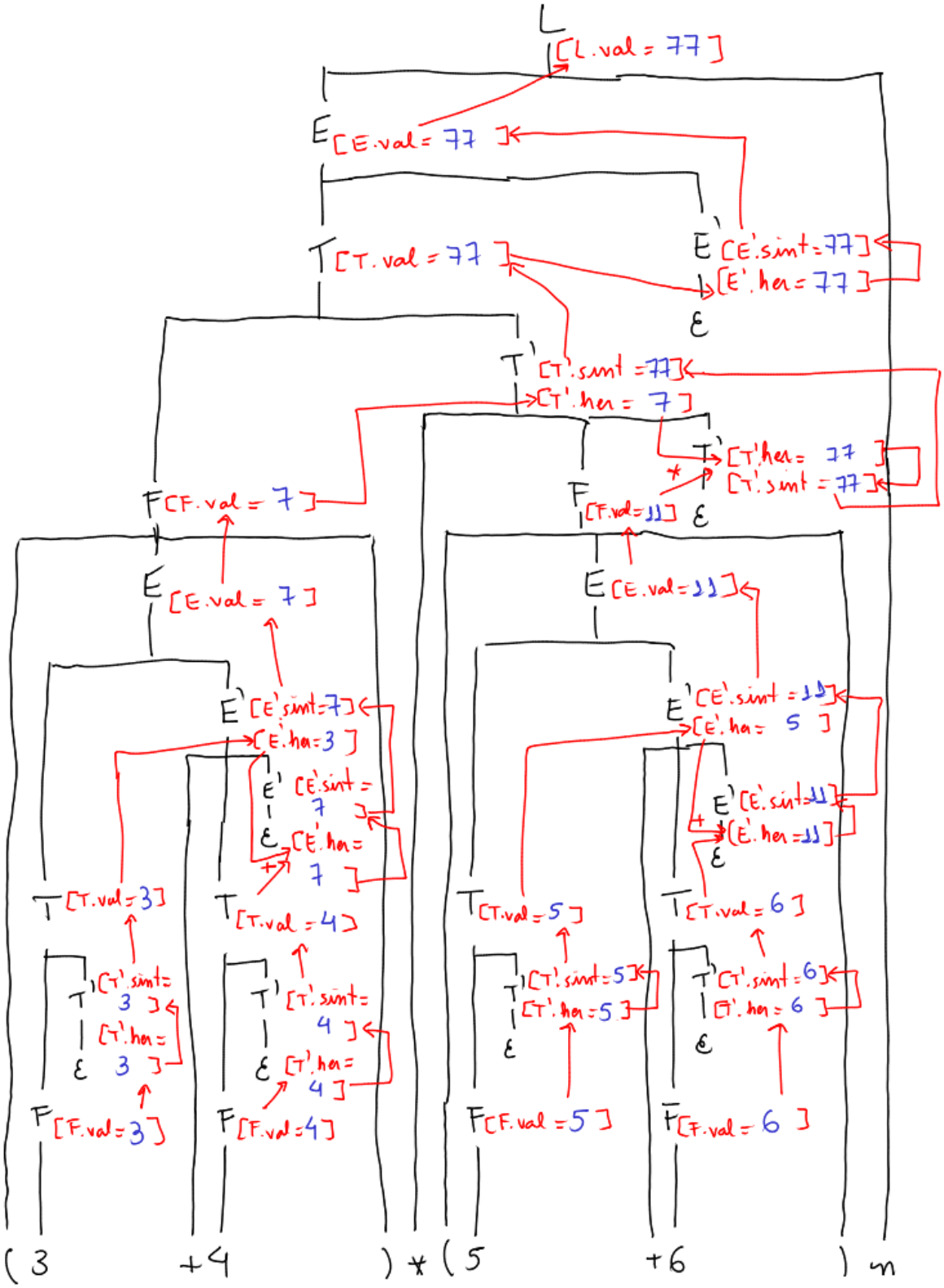
2) Estenda a DDS abaixo de forma a tratar expressões completas como no exercício 1

Produção	Regras semânticas
T → F T′	T′.her = F.val
	T.val = T′.sint
T′ → * F T′1	T′1.her = T′.her * F.val
	T′.sint = T′1.sint
T′ → ε	T′.sint = T′.her
F → dígito	F.val = dígito.lexval

R:

Produção	Regras semânticas
L → E n	L.val = E.val
E → T E′	E′.her = T.val
	E.val = E′.sint
E′ → + T E′1	E′1.her = E′.her + T.val
	E′.sint = E′1.sint
E′ → ε	E′.sint = E′.her
T → F T′	T′.her = F.val
	T.val = T′.sint
T′ → * F T′1	T′1.her = T′.her * F.val
	T′.sint = T′1.sint
T′ → ε	T′.sint = T′.her
F → (E)	F.val = E.val
F → dígito	F.val = dígito.lexval

3) Repita a letra a) do exercício 1 para a sua DDS do exercício 2



4) Considere a seguinte gramática simples de declarações de variáveis como na sintaxe de C:

```
decl → tipo varlista
tipo → 'int' | 'float'
varlista → id ',' varlista | id
```

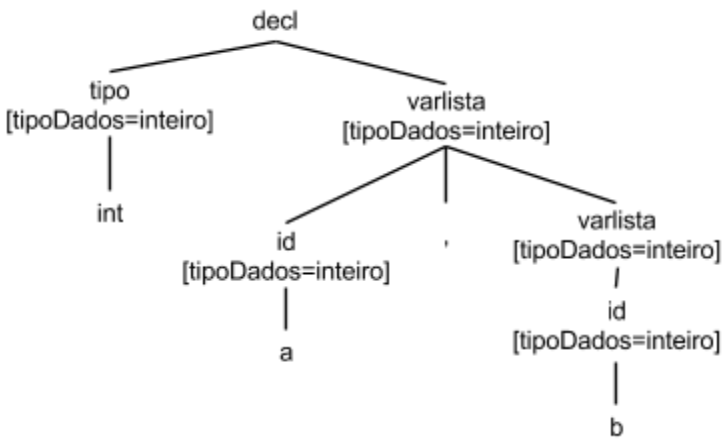
a) Construa a gramática de atributos para o atributo de tipo de dados, para o qual daremos o nome de tipoDados para diferenciar do não-terminal tipo

R.

Regra gramatical	Regras semânticas
decl → tipo varlista	varlista.tipoDados = tipo.tipoDados
tipo → 'int'	tipo.tipoDados = inteiro
tipo → 'float'	tipo.tipoDados = real
varlista → id ',' $varlista_1$	id.tipoDados = varlista.tipoDados $varlista_1$.tipoDados = varlista.tipoDados
varlista → id	id.tipoDados = varlista.tipoDados

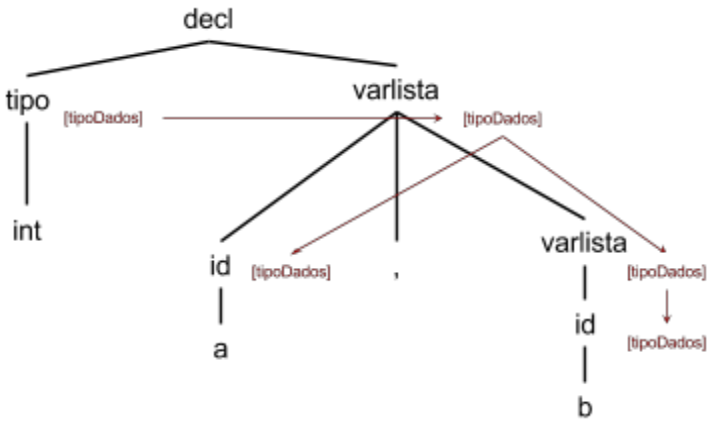
Observações:
– os valores de tipoDados pertencem ao conjunto {inteiro, real} o que corresponde aos tokens int e float
– o não-terminal tipo tem um tipoDados dado pelo token que ele representa
– esse tipoDados corresponde ao tipoDados de toda a varlista pela equação associada à regra gramatical para decl
– cada id na lista tem esse mesmo tipoDados pelas equações associadas a varlista
– veja que não há uma equação que envolva o tipoDados do não-terminal decl já que uma decl não precisa ter um tipoDados (não é necessário que o valor de um atributo seja especificado para todos os símbolos gramaticais)

b) Construa a árvore sintática com o cálculo dos atributos tipo_dados para a cadeia **`int a, b`**



c) Desenhe o grafo de dependência para a cadeia int a, b amarrado à árvore sintática construída na

letra (b)



d) Construa o esquema de TDS para a gramática do item (a)

R. (Obs: segundo a notação do ANTLR)

```
decl      : tipo varlista[$tipo.tipoDados]
          ;

tipo returns [ String tipoDados ]
          :      'int' { $tipo = "inteiro"; }
          ;

tipo returns [ String tipoDados ]
          :      'float' { $tipo = "real"; }
          ;

varlista [ String tipoDados ]
          :      ID ',' varlista [$tipoDados]
                 { definirTipo($ID.getText(),$tipoDados); }
          ;

varlista [ String tipoDados ]
          :      ID { definirTipo($ID.getText(),$tipoDados); }
          ;
```

Observações:
- ID é uma regra léxica, portanto tem o método getText() gerado pelo ANTLR
- Por ser uma regra léxica, ID teve seu nome renomeado para maiúsculas, conforme convenção do ANTLR
- O método definirTipo estabelece o tipo de uma variável, provavelmente inserindo na tabela de símbolos

5) O atributo tipoDados da questão 4 é sintetizado ou herdado? Por que?

R. É herdado, pois ele tem dependência de pai para filho e entre irmãos.

6) Escreva um procedimento recursivo para calcular o atributo tipoDados da questão 4, em todos os nós necessários.

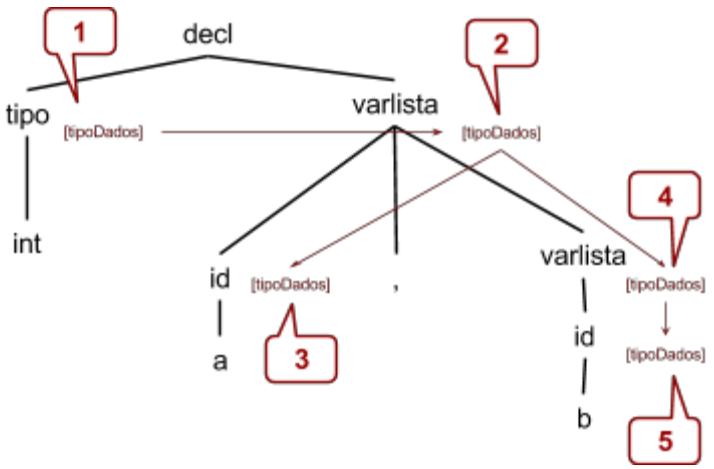
R.

```
procedure AvalTipo (T: noarvore);
begin
    case tipoNó de T of
    decl:
        AvalTipo (tipo filho de T);
        Atribui tipoDados de tipo filho de T a varlista filho de T;
        AvalTipo (varlista filho de T);
    tipo:
        if filho de T = int then T.tipoDados := inteiro
        else T.tipoDados = real;
    varlista:
        atribui T.tipoDados a primeiro filho de T;
        if terceiro filho de T não é nil then
            atribui T.tipoDados a terceiro filho;
            AvalTipo (terceiro filho de T);
    end case;
end AvalTipo;
```

Observe a mistura de pré-ordem e em-ordem dependendo do tipo distinto de nó processado. Por exemplo, um nó decl requer que o tipoDados de seu primeiro filho seja computado primeiro e, depois, atribuído ao segundo filho antes da chamada recursiva de AvalTipo naquele filho; esse processo é em-ordem. Um nó varlista, no entanto, atribui tipoDados aos seus filhos antes de qualquer chamada recursiva; isso é um processo em pré-ordem.

7) Enumere os nós da árvore construída na letra (c) da questão 4 indicando a ordem de computação de tipoDados de acordo com o algoritmo da questão 6. Que tipo de percurso é esse?

R. Combinação de percurso pré-ordem com em-ordem



8) Considere a seguinte gramática simples de expressões com uma única operação, a divisão (/), e dois tipos de operandos: números inteiros (sequências de dígitos indicados pelo token num) e números de ponto flutuante (indicados pela sequência num.num).

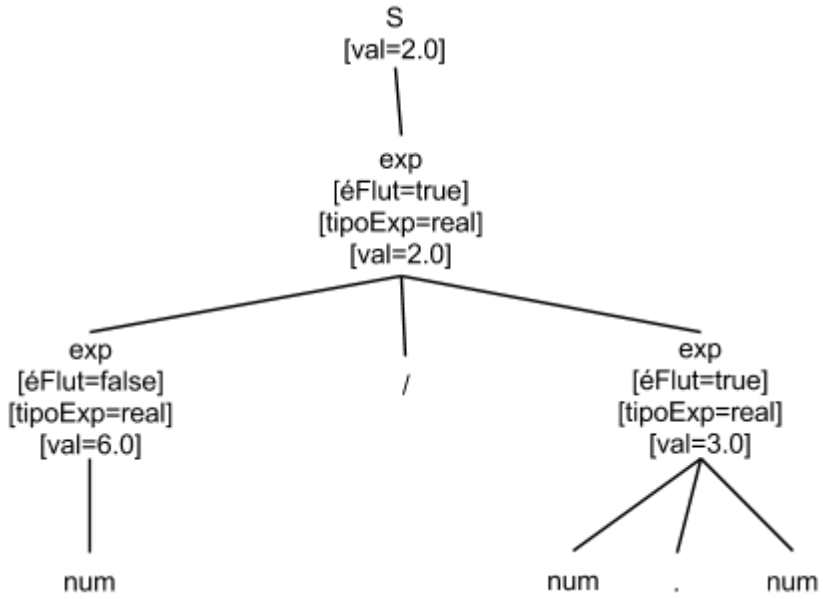
```
S → exp
exp → exp '/' exp | num | num '.' num
```

a) Construa uma gramática de atributos capaz de interpretar de maneira distinta a operação divisão dependendo do tipo de operandos envolvidos: se pelo menos um for ponto flutuante, a divisão será de ponto flutuante; se todos forem inteiros então a divisão será inteira. Três atributos deverão ser calculados: um que indica se a expressão é de ponto flutuante (**éFlut**), outro para o tipo da expressão (**tipoExp**) e um último para armazenar o valor da expressão (**val**).
Use **div** para divisão de inteira e **/** para divisão de ponto flutuante, assim 5/2.0 = 1,25 (5 / 2.0) e 5/2 = 2 (5 div 2).

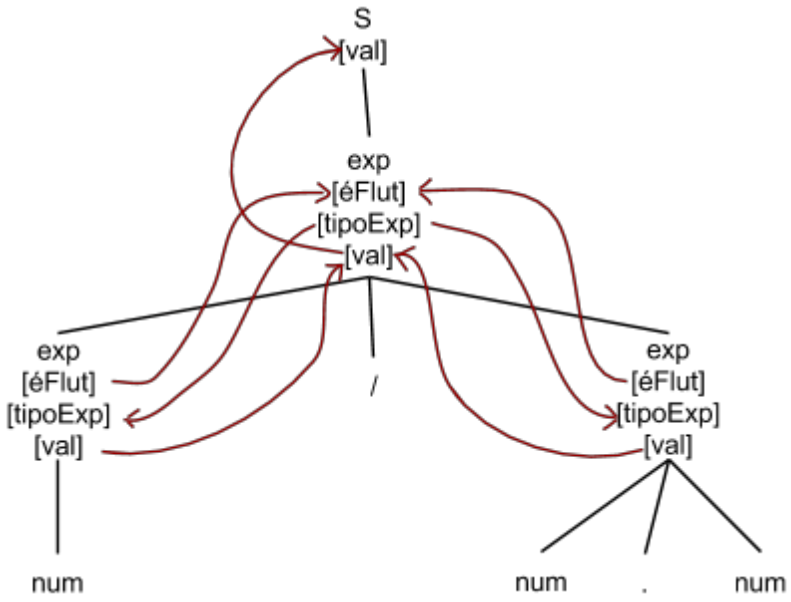
R.

Regra gramatical	Regras semânticas
$S \rightarrow exp$	exp.tipoExp = **if** exp.éFlut **then** real **else** inteiro S.val = exp.val
$exp_1 \rightarrow exp_2$ '/' exp_3	exp_1.éFlut = exp_2.éFlut **or** exp_3.éFlut exp_2.tipoExp = exp_1.tipoExp exp_3.tipoExp = exp_1.tipoExp exp_1.val = **if** exp_1.tipoExp == inteiro **then** exp_2.val **div** exp_3.val **else** exp_2.val **/** exp_3.val
$exp \rightarrow num$	exp.éFlut = false exp.val = **if** exp.tipoExp == inteiro **then** StringToInt(num.lexval) **else** StringToFloat(num.lexval)
$exp \rightarrow num_1$ '.' num_2	exp.éFlut = true exp.val = StringToFloat(num_1.lexval+"."+num_2.lexval)

b) Construa a árvore sintática com o cálculo dos atributos para a cadeia `6/3.0`



c) Desenhe o grafo de dependência para a cadeia **6/3.0** amarrado à árvore sintática construída na letra (b)



9) De que tipo (sintetizado ou herdado) é cada um dos 3 atributos calculados na questão 8? Justifique sua resposta.

R.

- **éFlut** é sintetizado, pois seu valor é calculado nas folhas (como mostram as regras semânticas dos casos base `exp → num`, para a qual éFlut é true; e `exp → num '.' num`, para a qual éFlut é false) e propagado para os pais.
- **val** também é sintetizado, pois seu valor é calculado nas folhas (como mostram as regras semânticas dos casos base `exp → num` e `exp → num '.' num` com base no valor numérico retornado pelo analisador léxico, lexval) e propagado para os pais.
- **tipoExp**, por sua vez, é herdado, pois seu valor é atribuído logo no símbolo inicial da gramática com base no valor de outro atributo, éFlut, e propagado para os nós filhos.

10) Descreva com palavras (não precisa fazer o algoritmo) como seria o processo para calcular os 3 atributos da questão 8. Quantas passadas seriam necessárias para calculá-los e qual o percurso usado nesse cálculo?

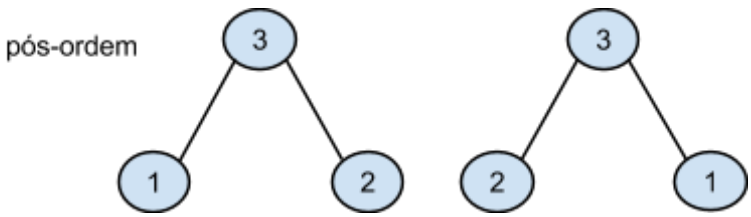
R. Os atributos **éFlut**, **tipoExp** e **val** podem ser computados em duas passadas na árvore sintática. A primeira passada computa o atributo sintetizado **éFlut** por um percurso em pós-ordem. A segunda passada computa o atributo herdado **tipoExp** e o atributo sintetizado **val** em um percurso combinado em pré-ordem e pós-ordem.

11) Diga quais são os dois tipos de gramáticas de atributos apresentados em aula explicando quais são as características de cada uma delas.

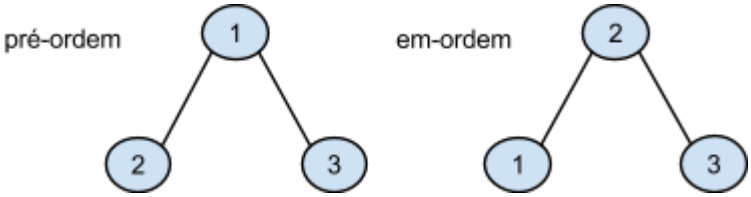
R. Os dois tipos são: Gramática S-atribuída e Gramática L-atribuída. Uma gramática S-atribuída é aquela que só possui atributos sintetizados, ou seja, para os quais os valores são computados exclusivamente a partir dos valores dos atributos filhos. Uma gramática L-atribuída, por sua vez, é aquela na qual a presença de atributos herdados é restringida para permitir que as ações semânticas possam ser executadas durante a análise sintática em uma única passada. Assim, em uma gramática L-atribuída, para um símbolo X no lado direito de uma regra de produção, a ação que calcula um atributo herdado de X deve aparecer à esquerda de X. Toda gramática S-atribuída é L-atribuída.

12) Considerando-se a árvore sintática para uma dada cadeia, diga quais são os percursos mais indicados nessa árvore para o cálculo de um atributo sintetizado e de um atributo herdado. Por que? Qual desses dois tipos de atributos é mais fácil de ser calculado, por que?

R.
- Um atributo sintetizado deve ser calculado usando um percurso em pós-ordem já que trata-se de um atributo para o qual o valor é calculado exclusivamente com base nos valores presentes em seus nós filhos e, portanto, os valores dos atributos dos filhos precisam ser conhecidos para se permitir o cálculo do atributo do nó pai. A ordem do cálculo dos atributos dos filhos é irrelevante.



- Um atributo herdado, por sua vez, é aquele para o qual o valor é calculado a partir dos valores dos atributos dos irmãos ou do pai e, nesse caso, o percurso mais indicado é o pré-ordem ou uma combinação de pré-ordem com em-ordem. Por exemplo, na ilustração abaixo, considerando-se que estamos calculando o valor do atributo herdado para o filho da direita, seguindo o percurso em pré-ordem ou em-ordem os valores de irmão e pai já seriam conhecidos.



Os atributos sintetizados são mais fáceis de serem calculados uma vez que se baseiam (a princípio) nas folhas (itens lexicais) e vão se propagando para a raiz. Dessa forma, qualquer percurso pós-ordem é suficiente para garantir o cálculo. Já os atributos herdados dependem de uma análise cuidadosa para se determinar a ordem de computação dos atributos dos filhos. E essa análise não pode ser feita de forma automática, pois é um problema intratável (problema da ordenação topológica).