

Prof. Dr. Daniel Leal Souza

Autômatos e Compiladores (EC8MA)

Compiladores (CC6MA)

daniel.leal.souza@gmail.com

daniel.souza@prof.cesupa.br

2024

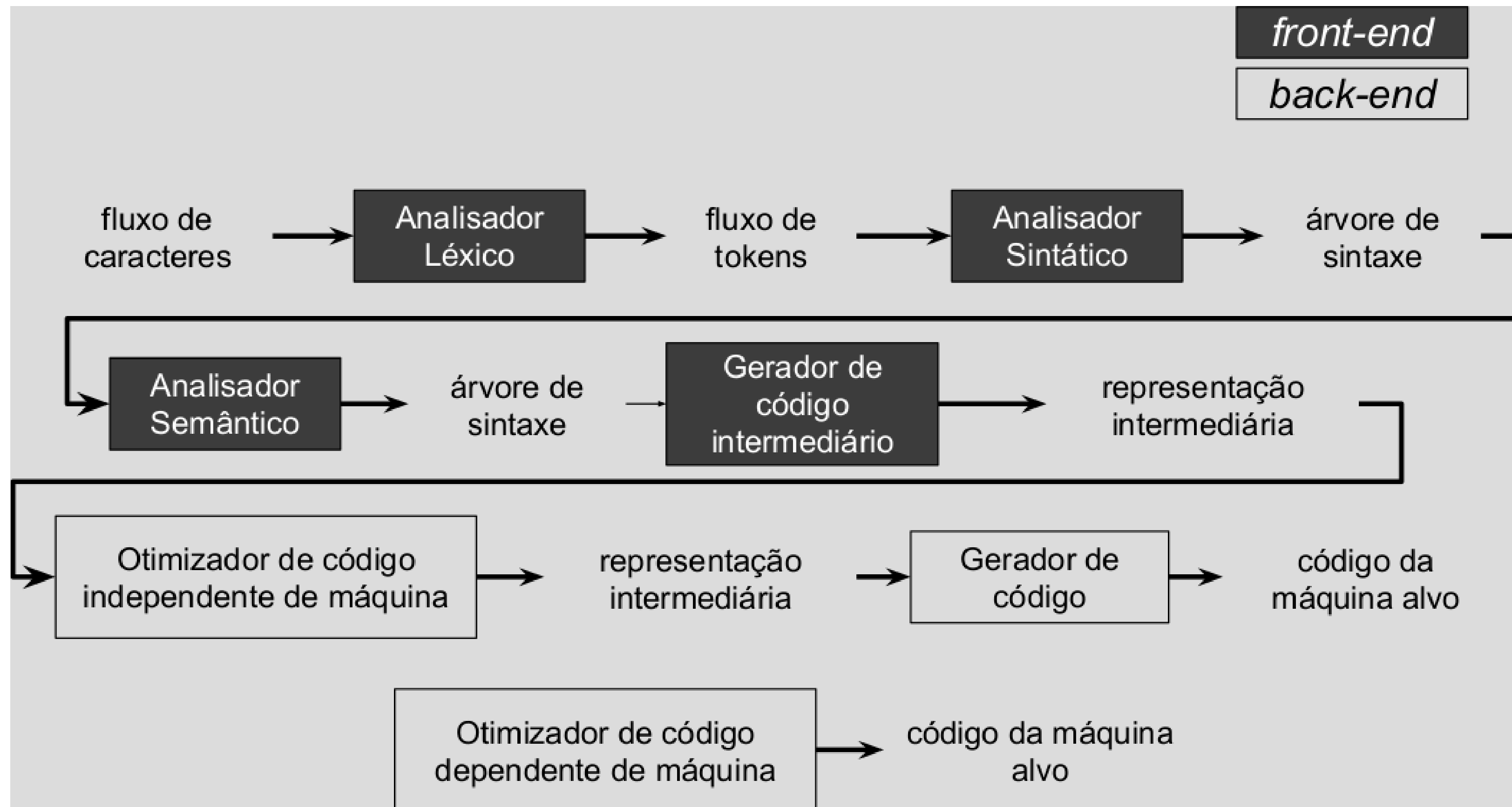


Estrutura de um compilador

- Duas etapas: análise e síntese

Análise (<i>front-end</i>)	Síntese (<i>back-end</i>)
<ul style="list-style-type: none">• Quebrar o programa fonte em partes;• Impor uma estrutura gramatical;• Criar uma representação intermediária;• Detectar e reportar erros (sintáticos e semânticos);• Criar a tabela de símbolos.	<ul style="list-style-type: none">• Construir o programa objeto com base na representação intermediária e na tabela de símbolos.

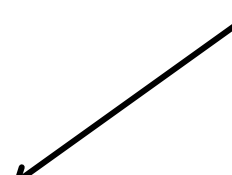
Fases de um compilador

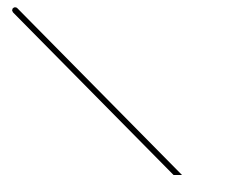


Fases de um compilador

- **Análise léxica (*scanning*)**

- Lê o fluxo de caracteres e os agrupa em sequências significativas
 - Chamadas lexemas
- Para cada lexema, um token é produzido
<nome-token, valor-atributo>

- 
- *Identifica o tipo do token*
 - *Símbolo abstrato, usado durante a análise sintática*

- 
- *Aponta para a tabela de símbolos (quando o token tem valor)*
 - *Necessária para análise semântica e geração de código*

Fases de um compilador

- **Exemplo: análise léxica**

position = initial + rate * 60

Geralmente,
espaços são
descartados.

Lexema	Token
position	<id,1>
=	<=>
initial	<id,2>
+	<+>
rate	<id,3>
*	<*>
60	<num,4>

entrada	lexema	tipo	...
1	position	int	...
2	initial	int	...
3	rate	float	...
4	60	int	...

Tabela de símbolos

Fases de um compilador

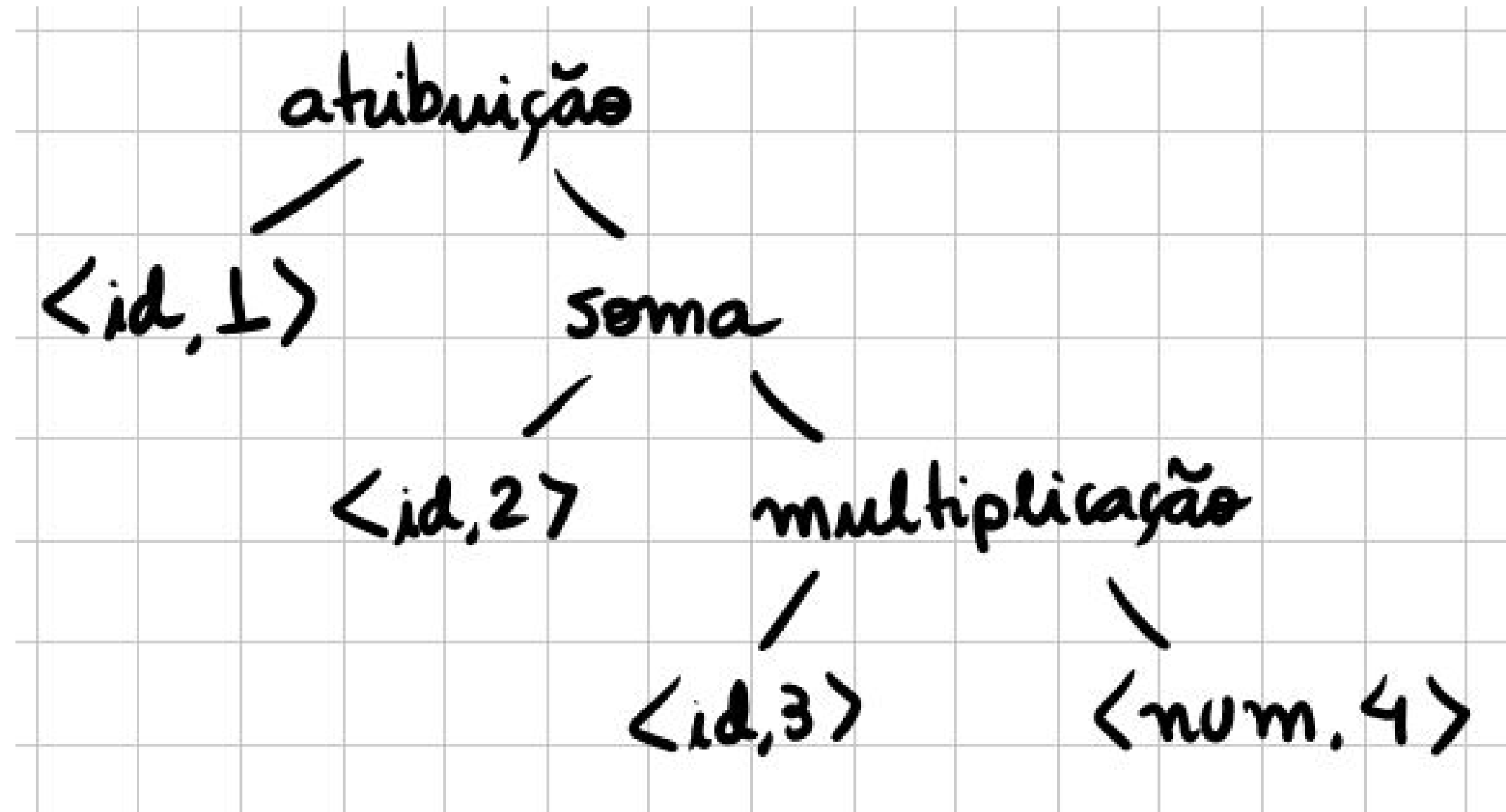
- **Análise sintática (*parsing*)**

- Usa os tokens produzidos pelo analisador léxico
 - Somente o primeiro “componente”
 - (ou seja, despreza os aspectos não-livres-de-contexto)
- Produz uma árvore de análise sintática
 - Representa a estrutura gramatical do fluxo de tokens
- As fases seguintes utilizam a estrutura gramatical para realizar outras análises e gerar o programa objeto

Fases de um compilador

- Exemplo: análise sintática

$\langle id, 1 \rangle \Rightarrow \langle id, 2 \rangle \langle + \rangle \langle id, 3 \rangle \langle * \rangle \langle num, 4 \rangle$



Fases de um compilador

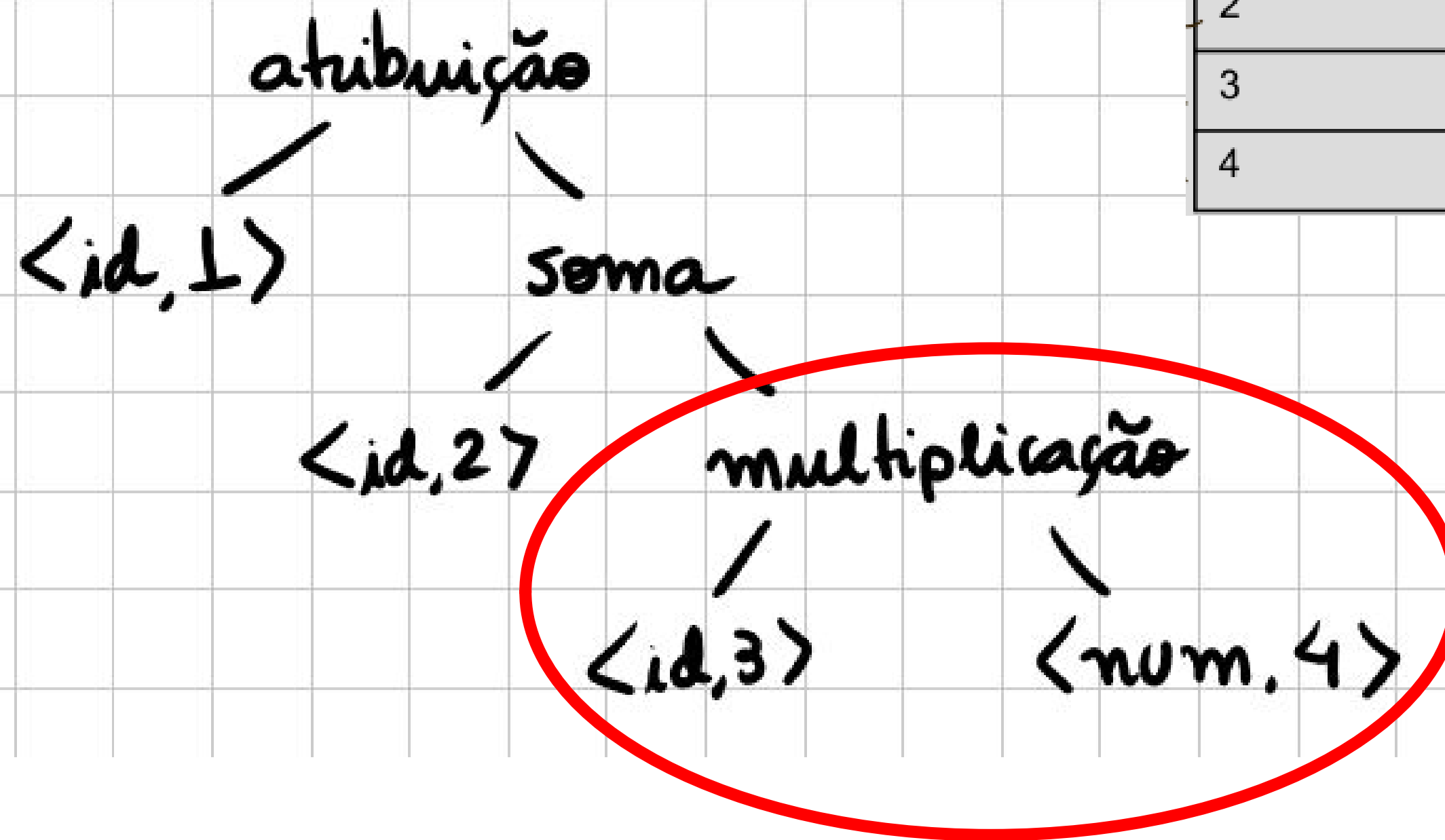
- **Análise semântica**

- Checa a consistência com a definição da linguagem;
- Coleta informações sobre tipos e armazena na árvore de sintaxe ou na tabela de símbolos;
- Checagem de tipos / coerção (adequação dos tipos) são tarefas típicas dessa fase.

Fases de um compilador

- Exemplo: análise semântica

entrada	lexema	tipo	...
1	position	int	...
2	initial	int	...
3	rate	float	...
4	60	int	...

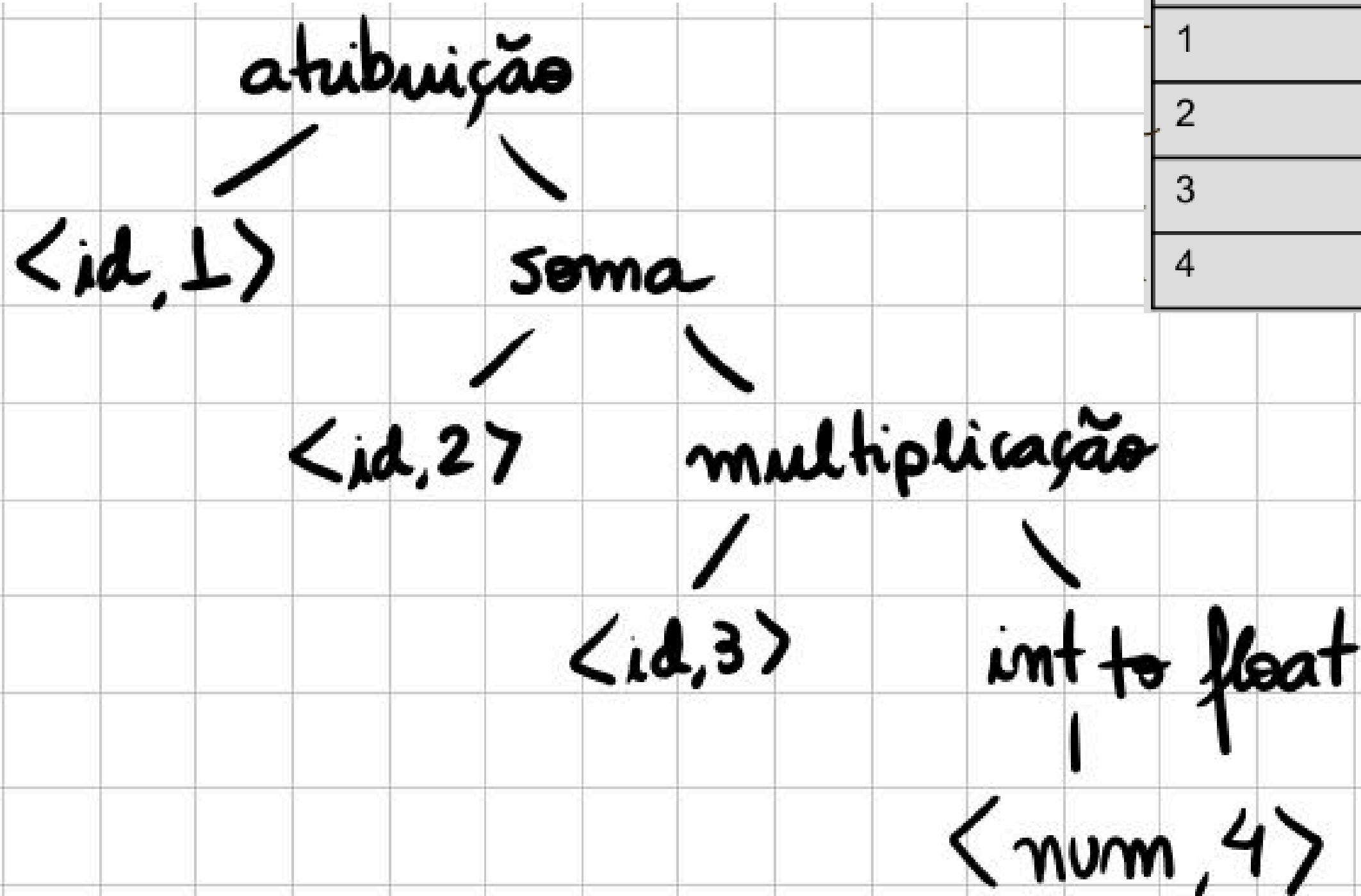


Tipos incompatíveis

Fases de um compilador

- Exemplo: coerção

entrada	lexema	tipo	...
1	position	int	...
2	initial	int	...
3	rate	float	...
4	60	int	...



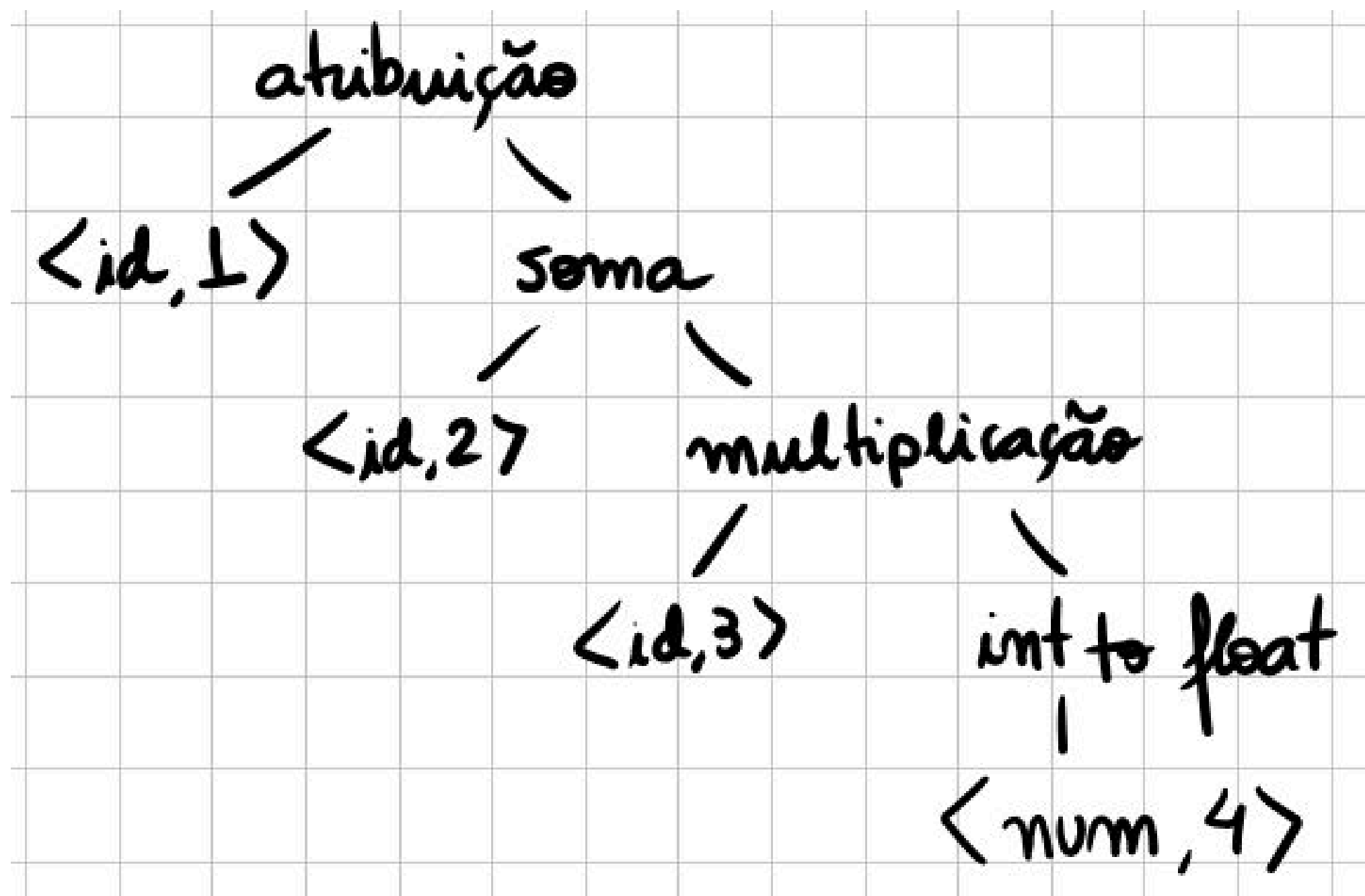
Fases de um compilador

- **Geração de código intermediário**

- Muitos compiladores geram uma representação intermediária, antes de gerar o código de máquina
- Motivos:
 - Fácil de produzir
 - Fácil de converter em linguagem de máquina
- Exemplos:
 - Árvore de sintaxe
 - Código de três endereços

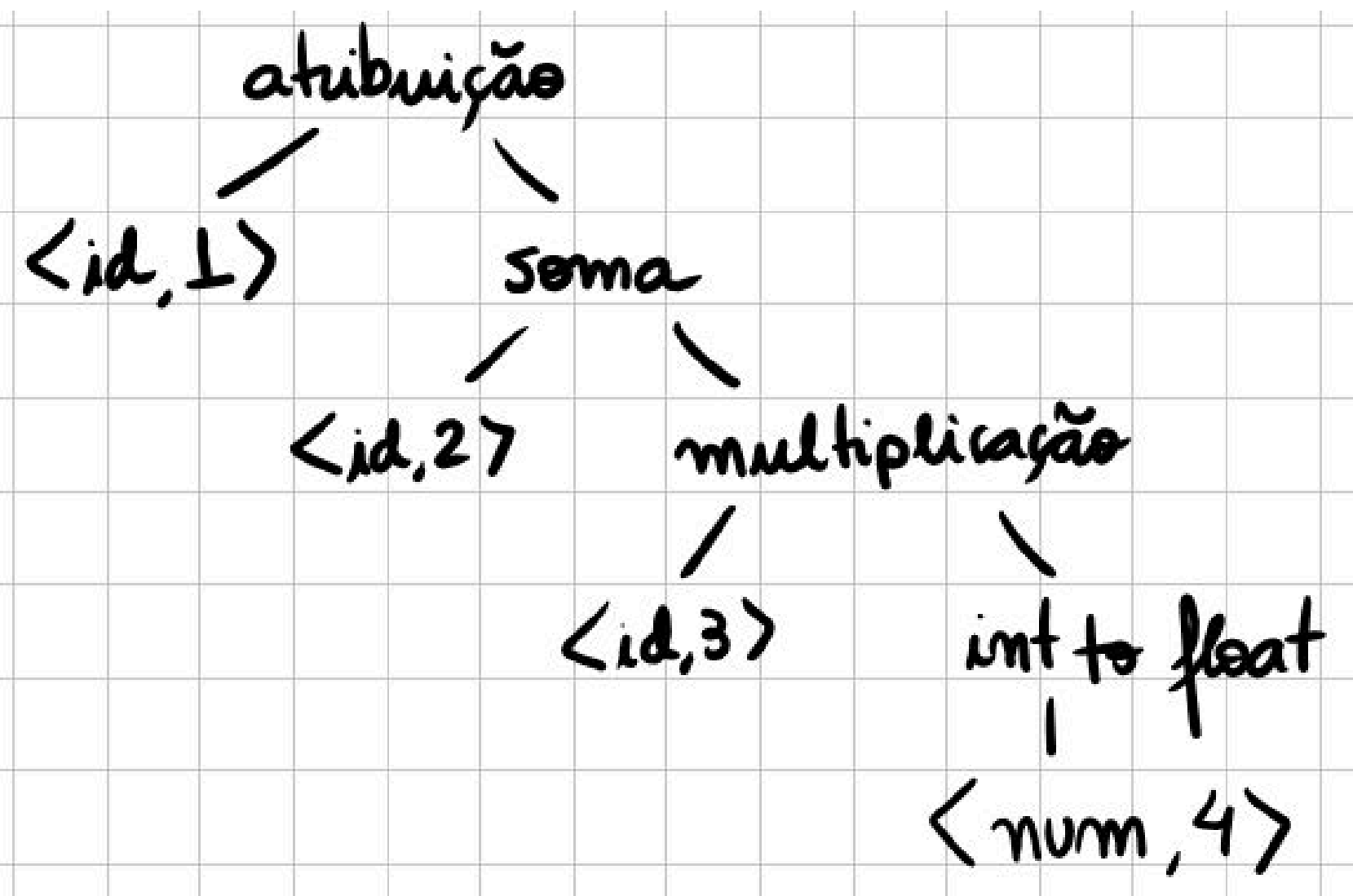
Fases de um compilador

- Exemplo: geração de código intermediário
 - Árvore de análise sintática já é um código intermediário



Fases de um compilador

- **Exemplo: geração de código intermediário**
 - Código de três endereços
 - Facilita geração de código objeto



```
num4 = 60
t1 = inttofloat(num4)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Fases de um compilador

- **Otimização de código**

- Tenta melhorar o código intermediário para produzir melhor código final
 - Mais rápido
 - Menor
 - Consome menos energia
 - Etc.
- Independentes x dependentes de máquina
- Quanto mais otimizações, mais lenta é a compilação
 - Porém, existem algumas otimizações simples, que levam a grandes melhorias

Fases de um compilador

- Otimização de código
 - Exemplo: otimização de código
 - Conversão “inttofloat” durante a compilação
 - Pode-se eliminar t3, pois é usado apenas uma vez

```
num4 = 60  
t1 = inttofloat(num4)  
t2 = id3 * t1  
t3 = id2 + t2  
id1 = t3
```

```
t1 = id3 * 60.0  
id1 = id2 + t1
```

Fases de um compilador

- **Exemplo: geração de código**
 - Código de máquina
 - Uso de registradores e instruções de máquina

t1 = id3 * 60.0
id1 = id2 + t1

LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1

Importante: é necessário lidar com endereços (não feito aqui)

Fases de um compilador

- **Gerenciamento da tabela de símbolos**
 - Fase “guarda-chuva”
 - Essencial: registrar nomes (variáveis, funções, classes, etc) usados no programa
 - Coletar informações sobre cada nome (tipo, armazenamento, escopo, etc)

DÚVIDAS?

Manipulação de erros

- Cada fase pode detectar diferentes erros
- Dependendo da gravidade, é possível que o compilador se “recupere” e continue lendo
 - Ou mesmo ignore o erro (ex: HTML)
- Em outros casos (na maioria), um erro desencadeia outros
- Mas, acredite, o compilador faz o melhor possível!

Manipulação de erros

```
int main()
{
    int i, a[1000000000000000];
    float j@;
    i = "1";
    while (i<3
        printf("%d\n", i);
    k = i;
    return (0);
}
```

Manipulação de erros

```
int main()
{
    int i, a[1000000000000000];
    float j@;
    i = "1";
    while (i<3
        printf("%d\n", i);
    k = i;
    return (0);
}
```

**Violação de tamanho
de memória:
Erro de geração de
código de máquina**

Manipulação de erros

```
int main()
{
    int i, a[1000000000000000];
    float j@;
    i = "1";
    while (i<3
        printf("%d\n", i);
    k = i;
    return (0);
}
```

**Violação de formação
de identificador:
Erro léxico**

Manipulação de erros

```
int main()
{
    int i, a[1000000000000000];
    float j@;
    i = "1";
    while (i<3
        printf("%d\n", i);
    k = i;
    return (0);
}
```

**Violação de
significado:
Erro semântico**

Manipulação de erros

```
int main()
{
    int i, a[1000000000000000];
    float j@;
    i = "1";
    while (i<3
        printf("%d\n", i);
    k = i;
    return (0);
}
```

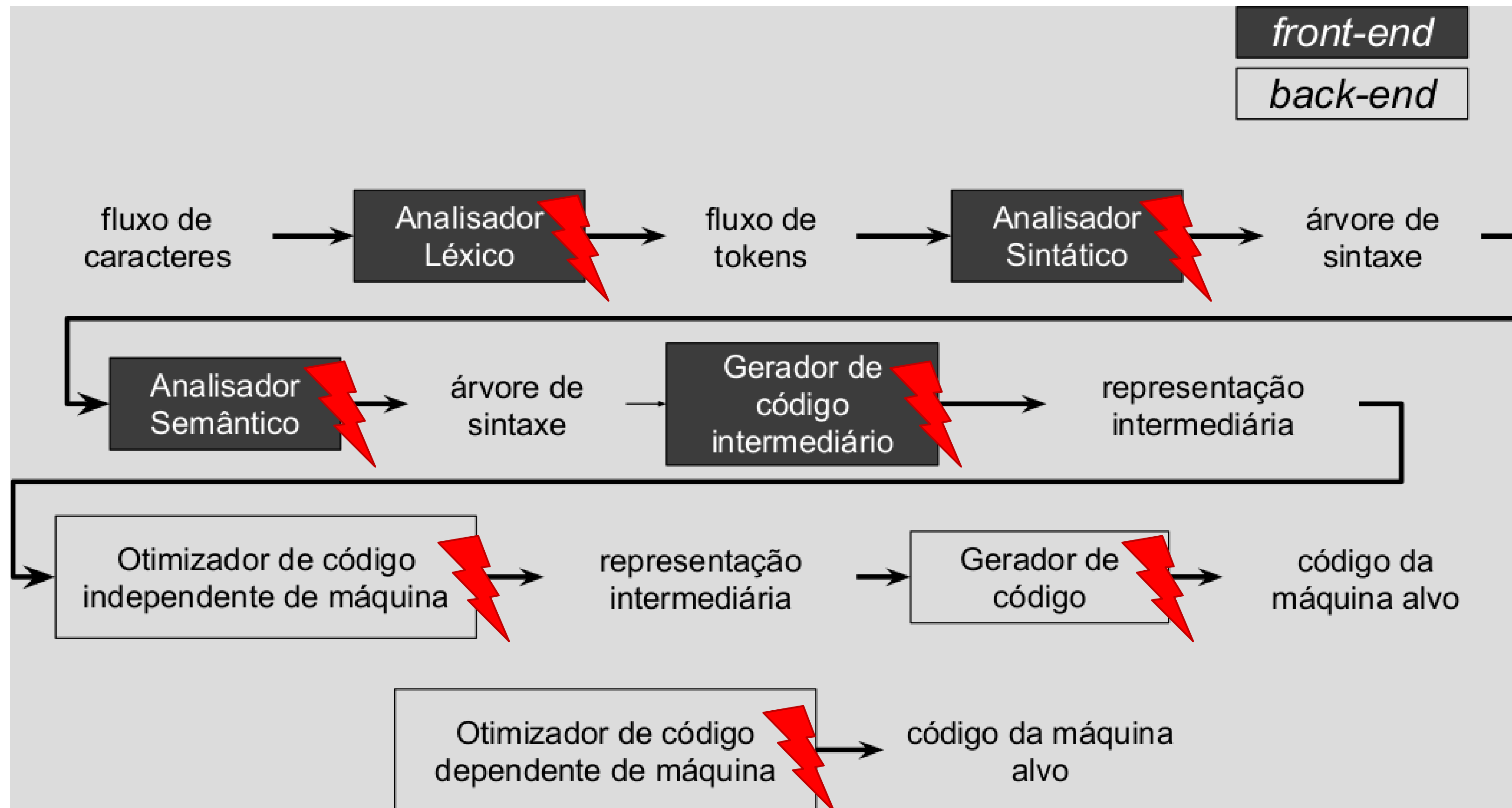
**Violação de formatação
de comando:
Erro sintático**

Manipulação de erros

```
int main()
{
    int i, a[1000000000000000];
    float j@;
    i = "1";
    while (i<3
        printf("%d\n", i);
    k = i;
    return (0);
}
```

**Violação de identificadores conhecidos (não declarado):
Erro contextual (“semântico”)**

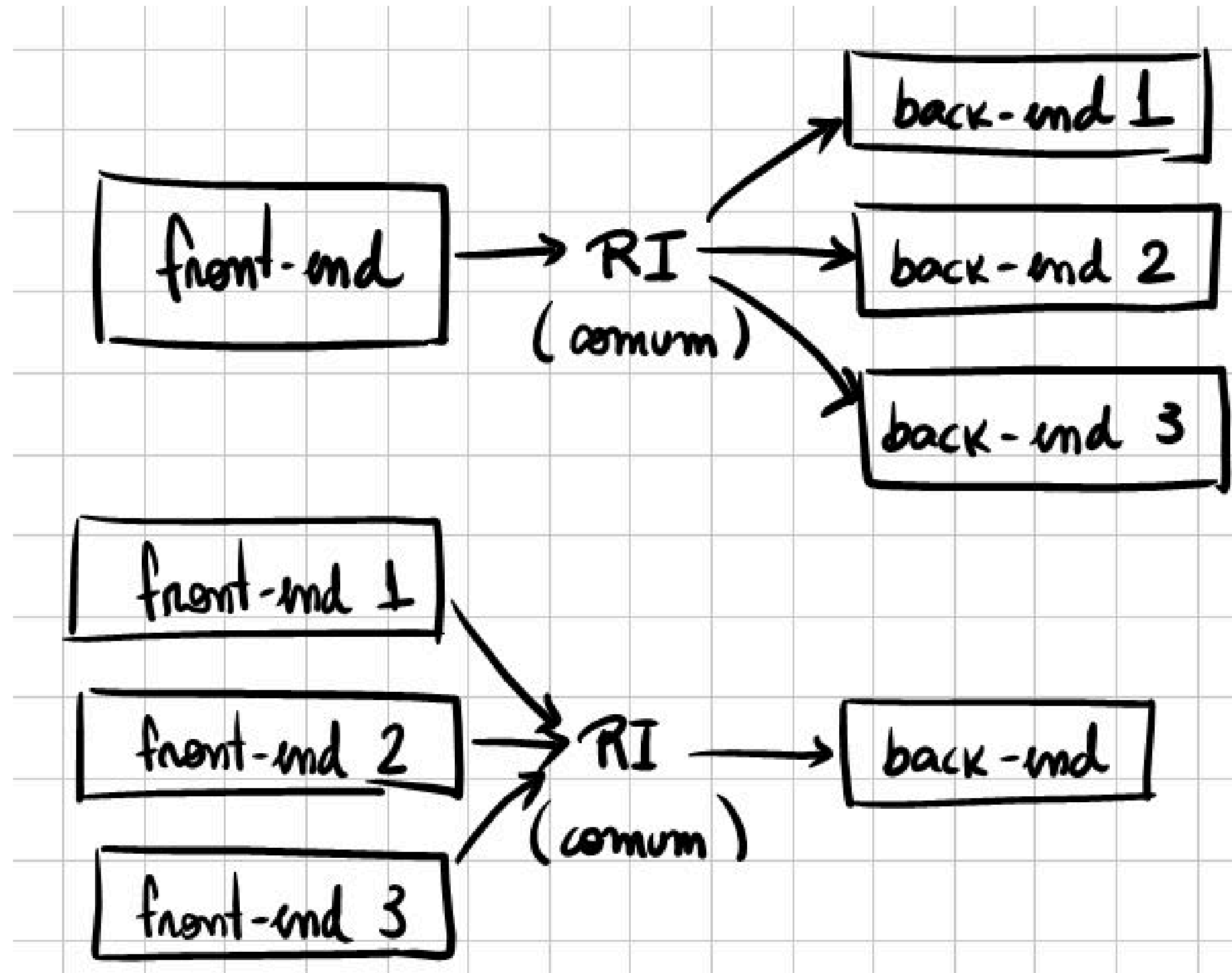
Manipulação de erros



Agrupamento das fases

- A divisão anterior é apenas lógica
- Pode-se realizar várias fases de uma única vez
- Em uma única passada
 - Imagine que o programa está numa fita VHS
 - Cada passada é um “play” na fita toda
 - Ao fim de cada passada, precisa rebobinar
- Exemplo:
 - Passada 1 = análise léxica, sintática, semântica e geração de representação intermediária (*front-end*)
 - Passada 2 = otimização (opcional)
 - Passada 3 = geração de código específico de máquina (*back-end*)

Agrupamento das fases



RI: Representação intermediária

- Nesta disciplina o foco estará no front-end
 - Análise léxica + sintática + semântica
 - Entretanto, vamos estudar um pouco de geração de código / interpretação também

Assuntos que são suficientes para muitas aplicações práticas interessantes!

DÚVIDAS?

Referências

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman.
Compiladores - princípios, técnicas e ferramentas. Pearson, 2007.
- José Neto, João. Apresentação à compilação. 2ª edição. Rio de Janeiro: Elsevier, 2016.
- Notas de aula do professor Daniel Lucrédio - UFSCar.