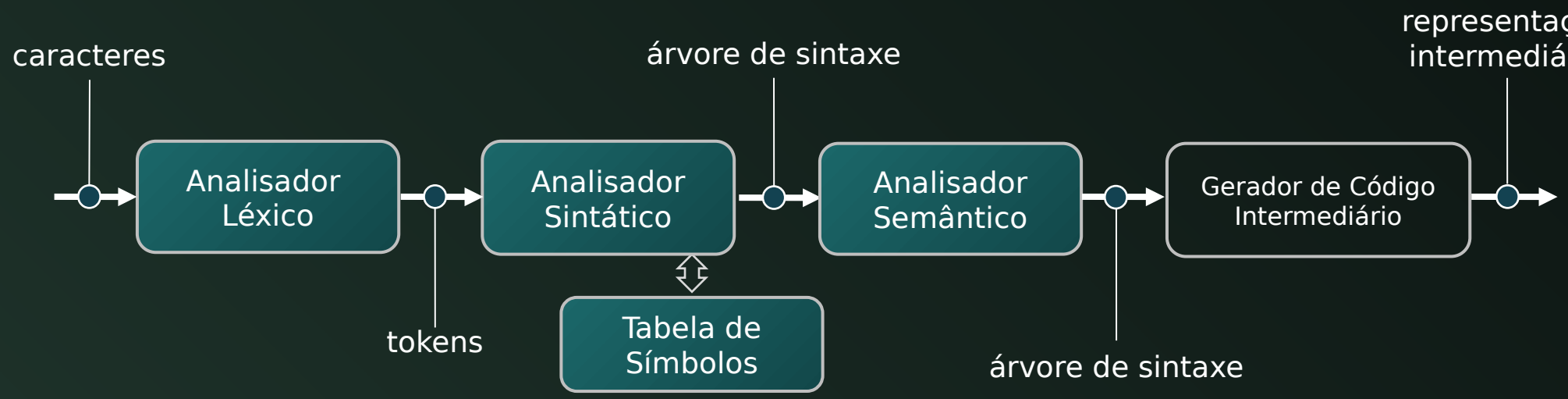
Judson Santos Santiago

Geração de Código Intermediário

Compiladores

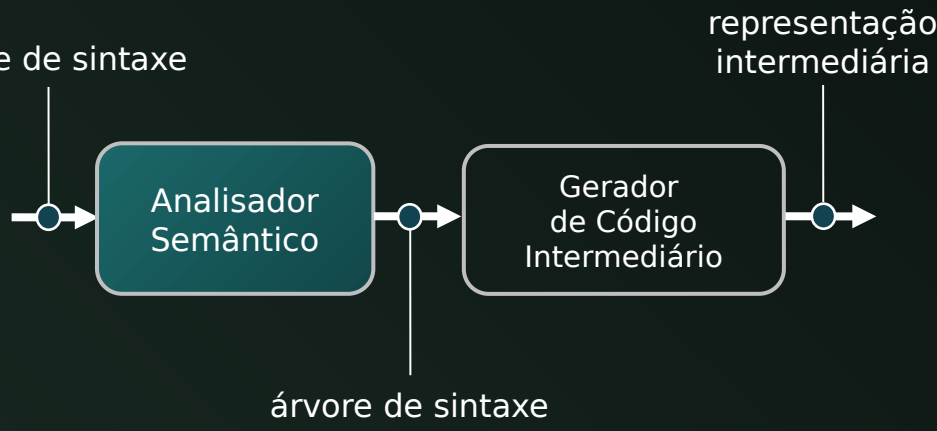
Introdução

- O **processador de linguagem** desenvolvido até o momento faz:
  - Análise léxica, sintática e semântica
  - Uso da tabela de símbolos



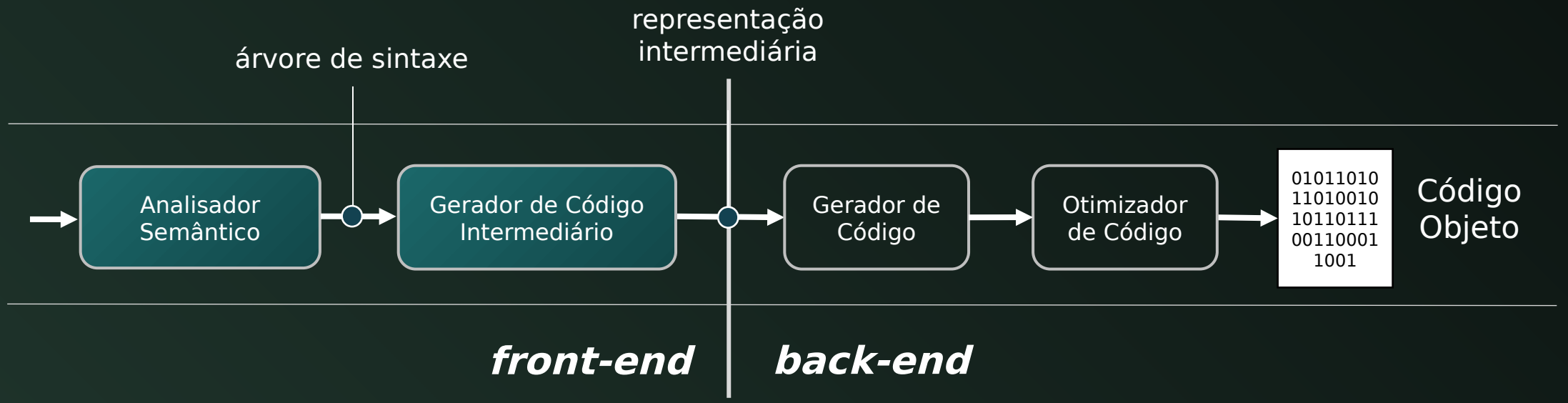
Introdução

- O *front-end* de um compilador verifica se um programa segue as regras sintáticas e semânticas da linguagem
  - Para isso ele constrói uma representação do código fonte
- As mais importantes são:
  - Árvores
    Árvore de sintaxe abstrata
  - Representações lineares
    Código de três endereços



Introdução

- As representações lineares atuam como representações intermediárias entre o *front-end* e o *back-end*



Código de Três Endereços

- O código de três endereços:
  - É uma representação intermediária do código fonte
  - Pode ser gerado percorrendo a árvore de sintaxe
  - Constituído por uma sequência de instruções:
    - Atribuição
    - Desvio
    - Repetição
  - Suporta expressões com:
    - Operadores (ARI, REL, LOG)
    - Arranjos

```
x = y op z
```

Cada instrução trabalha com no máximo 3 endereços de memória

Código de Três Endereços

- As instruções e expressões em código de três endereços tem o formato abaixo:

  - Atribuição

    ```
    x = y
    ```

  - Operações

    ```
    x = y op z
    ```

  - Arranjos

    ```
    x[y] = z
    x = y[z]
    ```

Sendo x, y e z identificadores ou **temporários** gerados pelo compilador

Código de Três Endereços

- Atribuições e expressões mais complexas precisam ser decompostas em partes mais simples para não ultrapassar o limite de três endereços

código fonte

```
x = y * z + w
```

código de
três endereços

```
t1 = y * z
t2 = t1 + w
x = t2
```

Nomes temporários
precisam ser gerados
pelo compilador

Código de Três Endereços

- As instruções de três endereços são **executadas em sequência**
  - A não ser que um desvio condicional (ou incondicional) seja forçado

```
ifFalse x goto L
ifTrue x goto L
goto L
```

Instruções de desvio

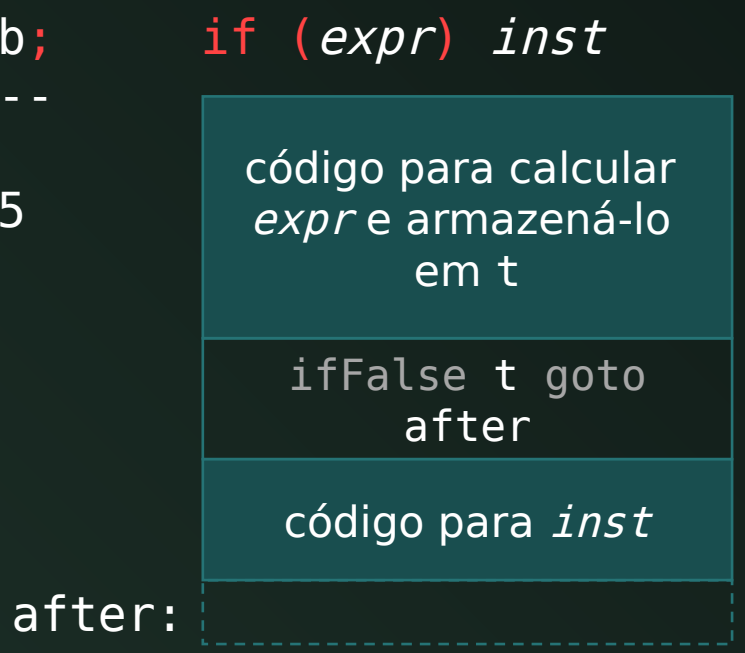
  - Um rótulo L pode ser conectado a qualquer instrução
    - Uma instrução pode ter mais de um rótulo

```
      1: i = 0
      2: j = 1
soma: 3: t = i + j
```

Código de Três Endereços

- As construções complexas são traduzidas usando desvios
  - O `if` é implementado usando o seguinte fluxo de controle

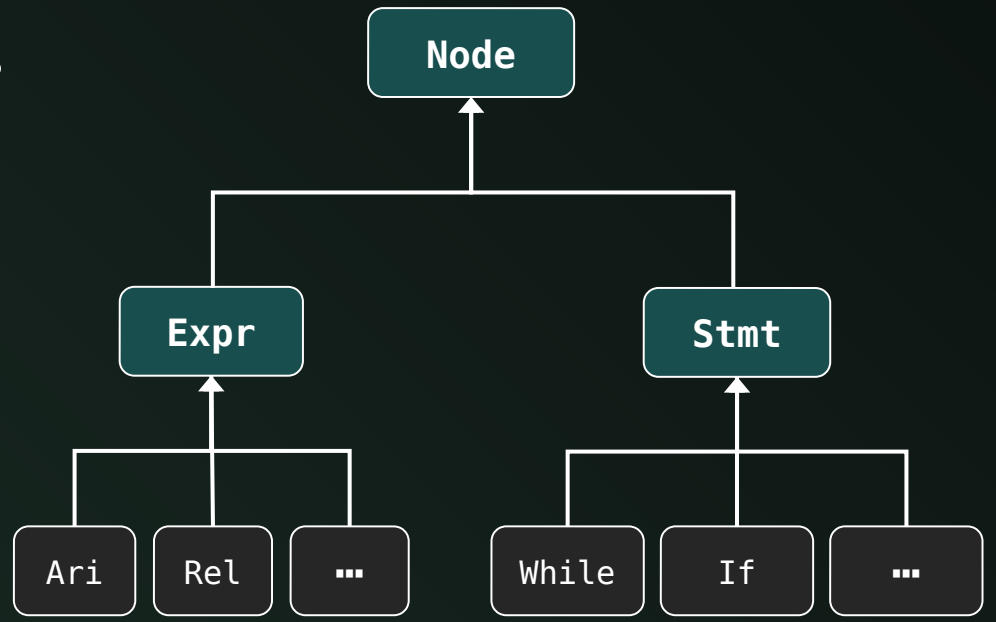
```
if (a > 2) c = a + b;
----------------------
1: t1 = a > 2
2: ifFalse t1 goto 5
3: t2 = a + b
4: c = t2
5:
```



Outras construções são traduzidas de forma similar

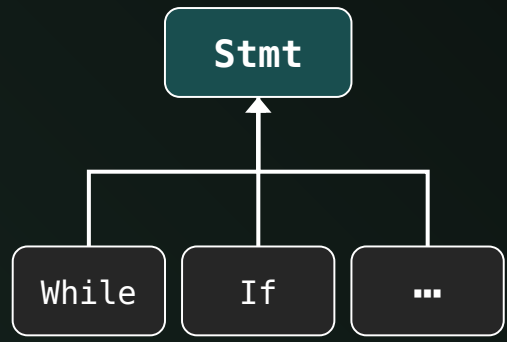
Geração de Código Intermediário

- A geração é feita a partir da árvore de sintaxe abstrata
  - É preciso gerar código de três endereços para cada nó da árvore sintática que represente uma **instrução** ou **expressão**
  - Nenhum código é gerado para declarações de variáveis e constantes

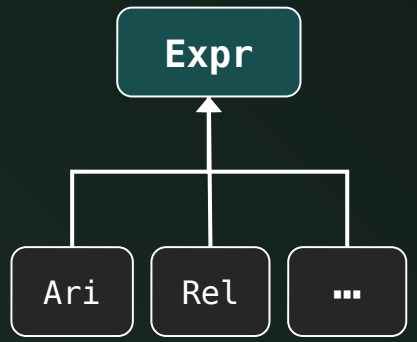


Geração de Código Intermediário

- Para instruções:
  - Cada classe derivada de `Stmt` possui um construtor e um método `Gen()` para gerar código de três endereços



- Para expressões:
  - São utilizadas duas funções chamadas de `Lvalue()` e `Rvalue()`



Geração de Código Intermediário

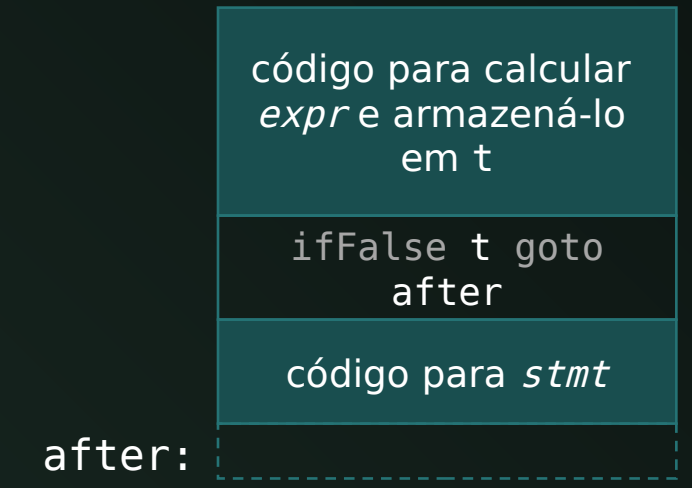
- A geração de código para **instruções**

```
class If : public Stmt
{
private:
    Expr E;
    Stmt S;

public:
    If(Expr e, Stmt s) : E(e), S(s) { after = NewLabel(); }
    void Gen()
    {
       Expr n = E.Rvalue();
       Print("ifFalse " + n.ToString() + " goto " + after);
       S.Gen();
       Print(after + ":");
    }
}
```

```
if (a > 2) c = a + b;
--------------------
1: t1 = a > 2
2: ifFalse t1 goto 5
3: t2 = a + b
4: c = t2
5:
```

if (*expr*) *stmt*



Geração de Código Intermediário

- A geração de código para **expressões**

  - Usa funções que quando aplicadas a um nó n:
    - **`Rvalue()`**: gera instruções para calcular n em um temporário e retorna um novo nó representando o temporário

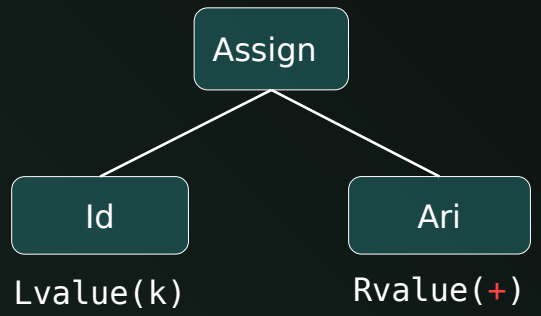
      ```
      t1 = i + j
      ```

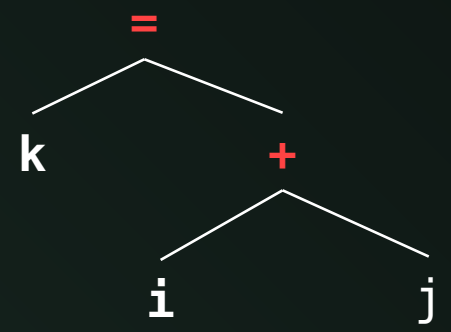
    - **`Lvalue()`**: gera instruções para traduzir as subárvores abaixo de n e retorna um nó representando o "local de armazenamento" para n

      ```
      k = t1
      ```

Árvore sintática

`k = i + j`





Geração de Código Intermediário

- Em acessos a arranjos é preciso distinguir entre valores-l e valores-r
  - A expressão 2*a[i] pode ser traduzida em código de três endereços colocando a[i] em um temporário

```
t1 = a[i]
t2 = 2 * t1
```

  - No entanto se a[i] aparecer no lado esquerdo da expressão não se pode usar um temporário no lugar do arranjo

```
a[i] = 2 * k
```

Geração de Código Intermediário

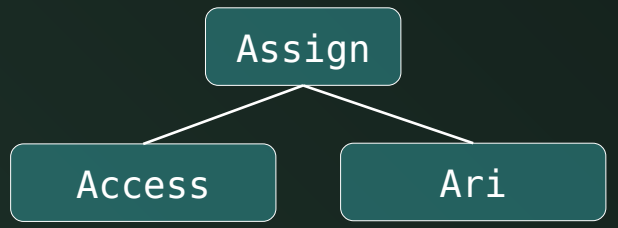
- A função **Lvalue**:
  - Retorna n se ele for um simples identificador
  - Retorna um novo nó Access se ele for um arranjo

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}
```

se n representa a[2*k],
a função gera t = 2*k e
retorna um novo nó
para a[t]

Geração de Código Intermediário

- A função **Rvalue**



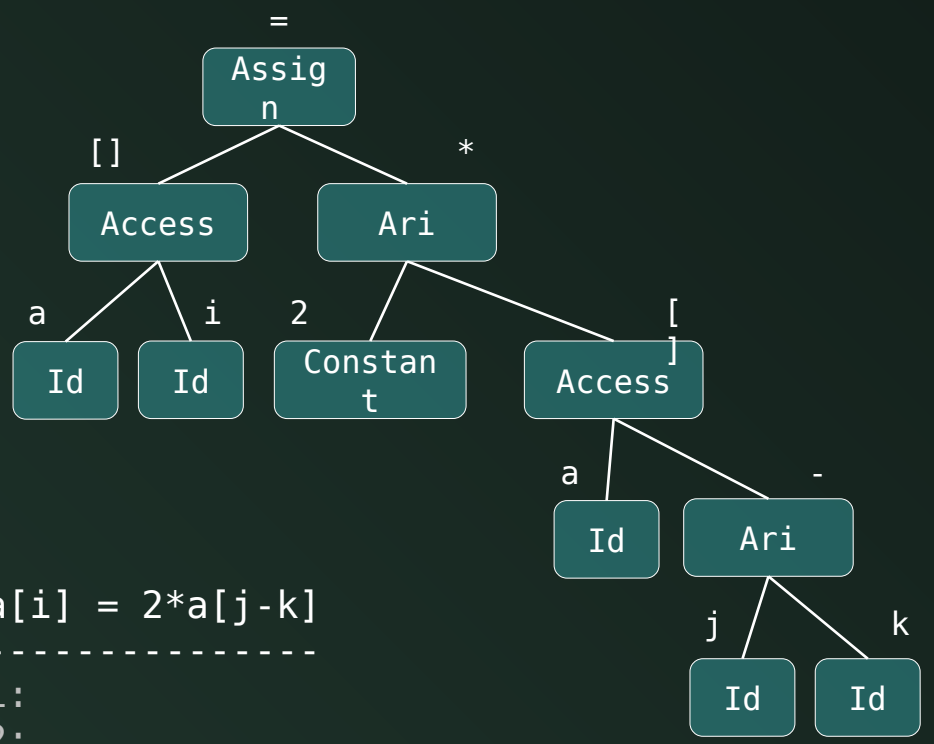
a[i] = 2*a[j-k]

O método Gen() de Assign vai emitir a string:

Lvalue([]) "=" Rvalue(*)

```
Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
    else if (n é um nó Assign(y,z)) {
        emite string para Lvalue(y) = Rvalue(z);
        return z';
    }
}
```

Exemplo de Geração

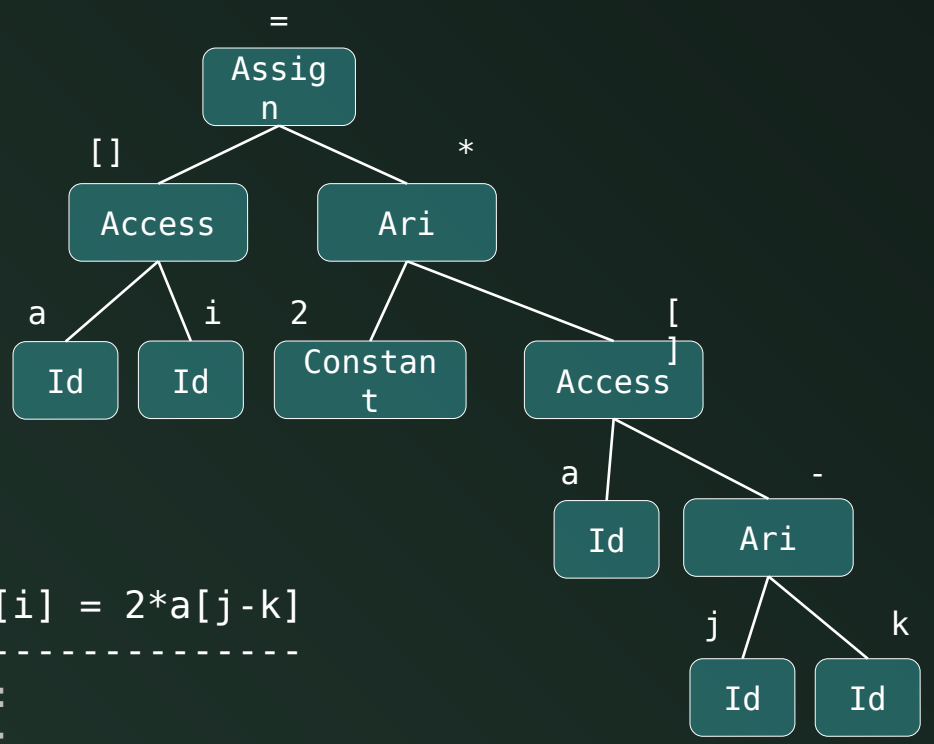



```
a[i] = 2*a[j-k]
---------------
1:
2:
3:
4: Lvalue([]) = Rvalue(*)
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

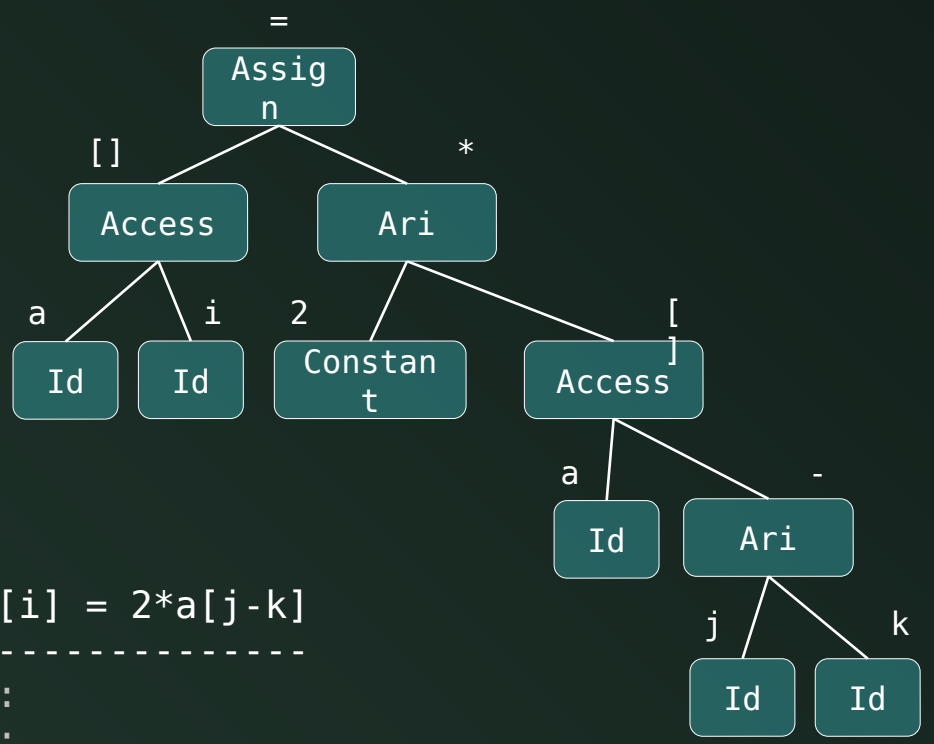



```
a[i] = 2*a[j-k]
---------------
1:
2:
3:
4: Access(a,Rvalue(i)) = Rvalue(*)
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

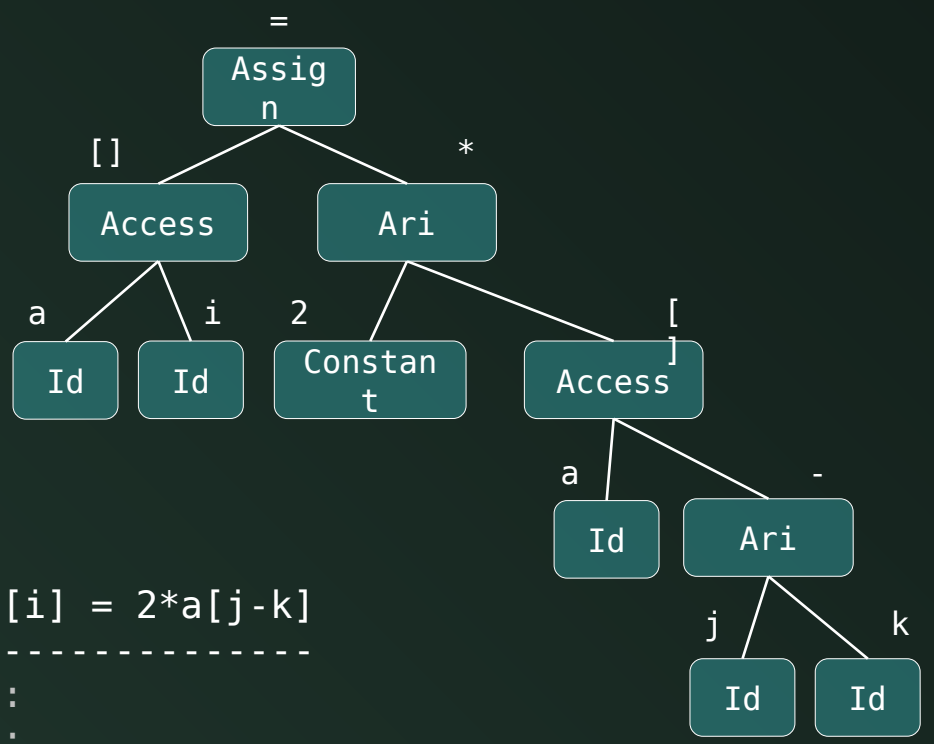



```
a[i] = 2*a[j-k]
---------------
1:
2:
3:
4: Access(a,i) = Rvalue(*)
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

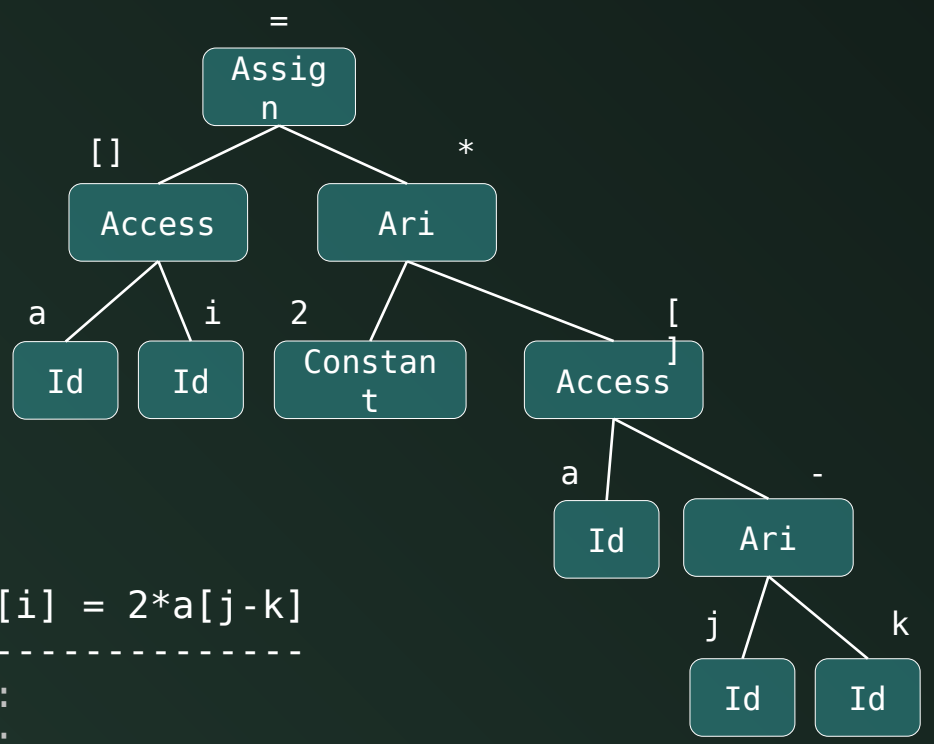



```
a[i] = 2*a[j-k]
---------------
1:
2:
3:
4: a[i] = Rvalue(*)
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

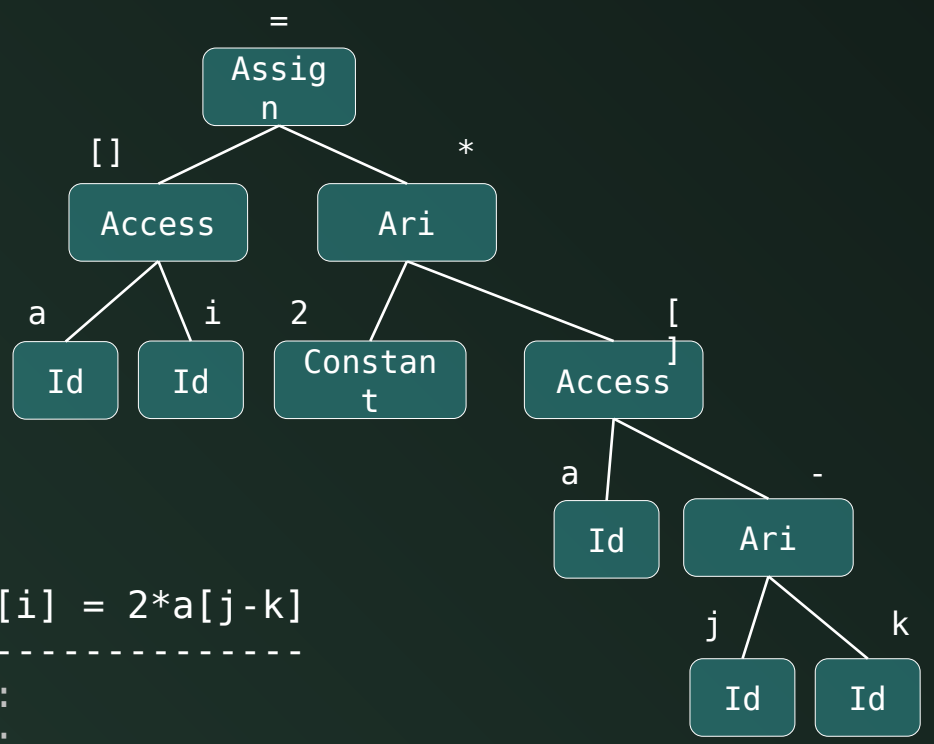



```
a[i] = 2*a[j-k]
---------------
1:
2:
3: t1 = Rvalue(2) * Rvalue([])
4: a[i] = Rvalue(*)
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

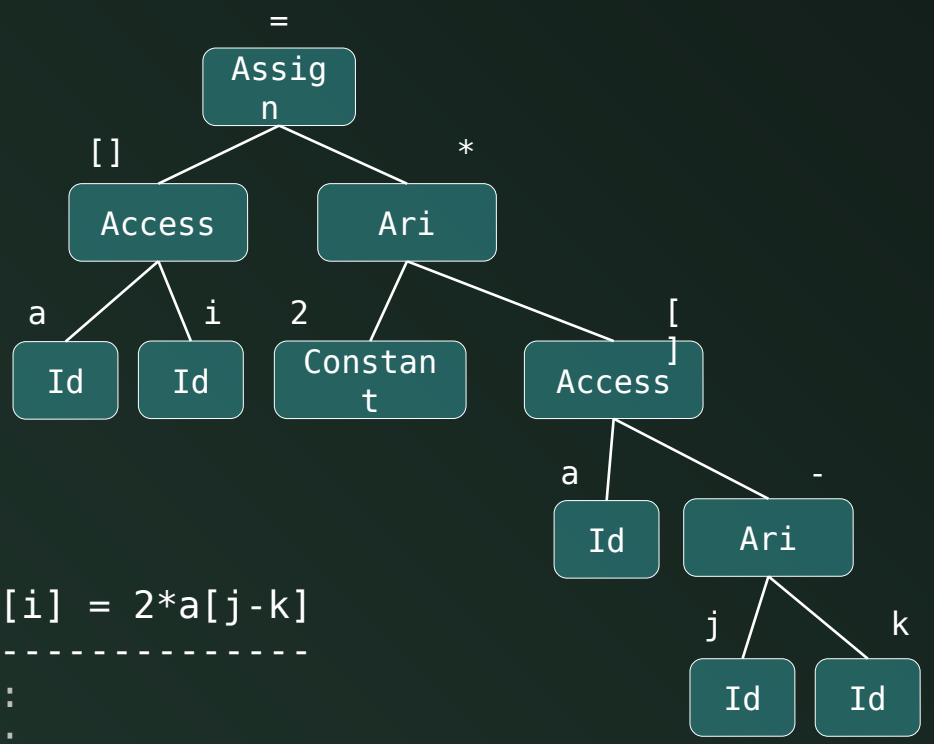



```
a[i] = 2*a[j-k]
---------------
1:
2:
3: t1 = Rvalue(2) * Rvalue([])
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

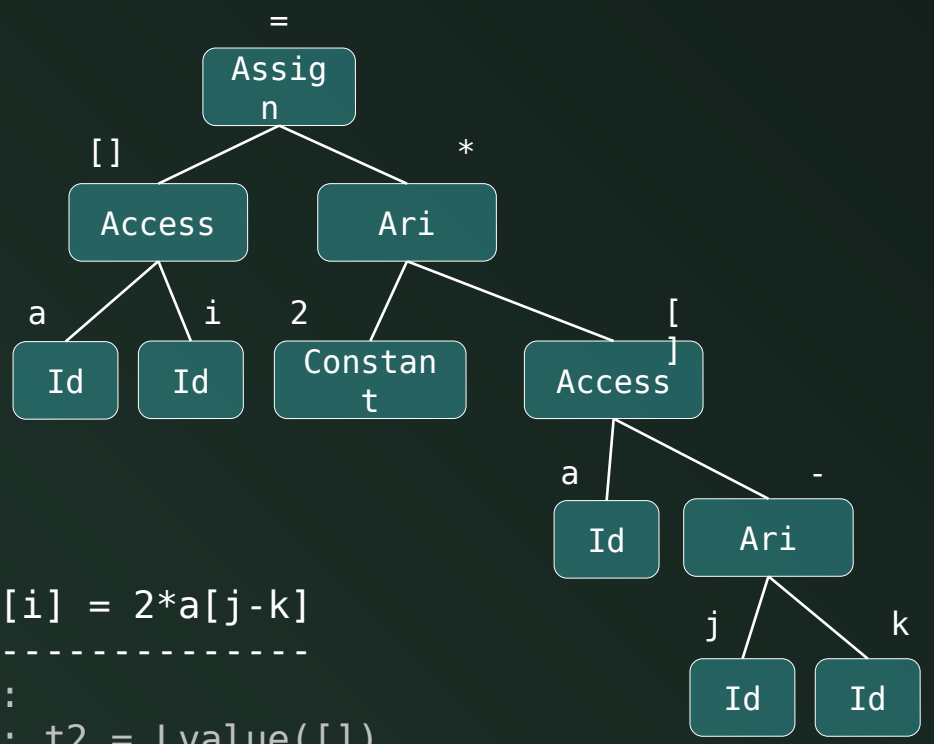



```
a[i] = 2*a[j-k]
---------------
1:
2:
3: t1 = 2 * Rvalue([])
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

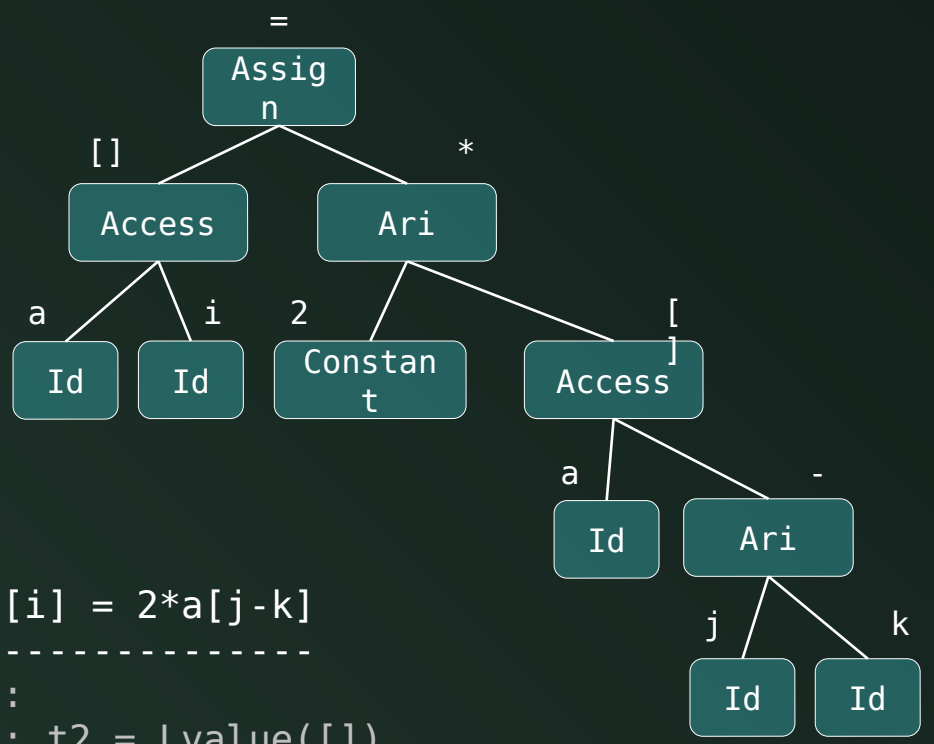



```
a[i] = 2*a[j-k]
---------------
1:
2: t2 = Lvalue([])
3: t1 = 2 * Rvalue([])
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

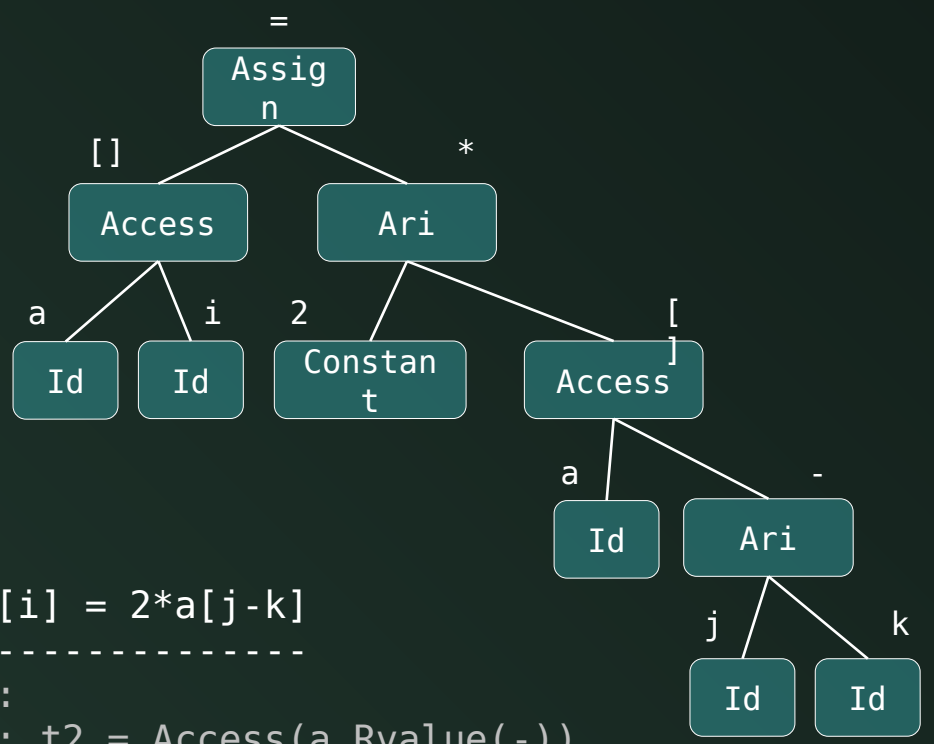



```
a[i] = 2*a[j-k]
---------------
1:
2: t2 = Lvalue([])
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

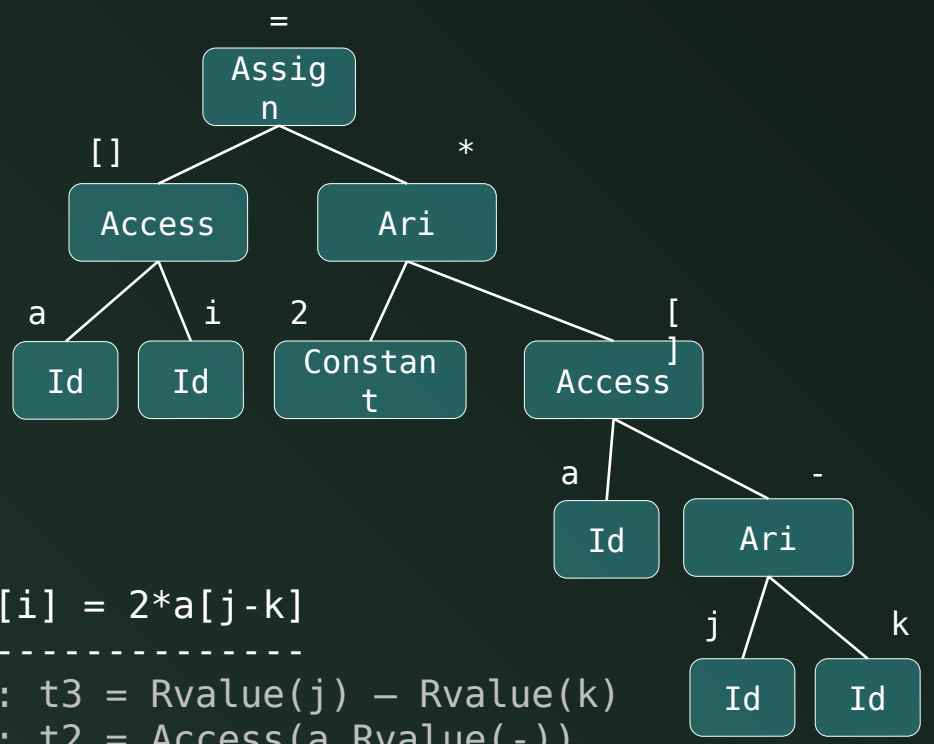



```
a[i] = 2*a[j-k]
---------------
1:
2: t2 = Access(a,Rvalue(-))
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

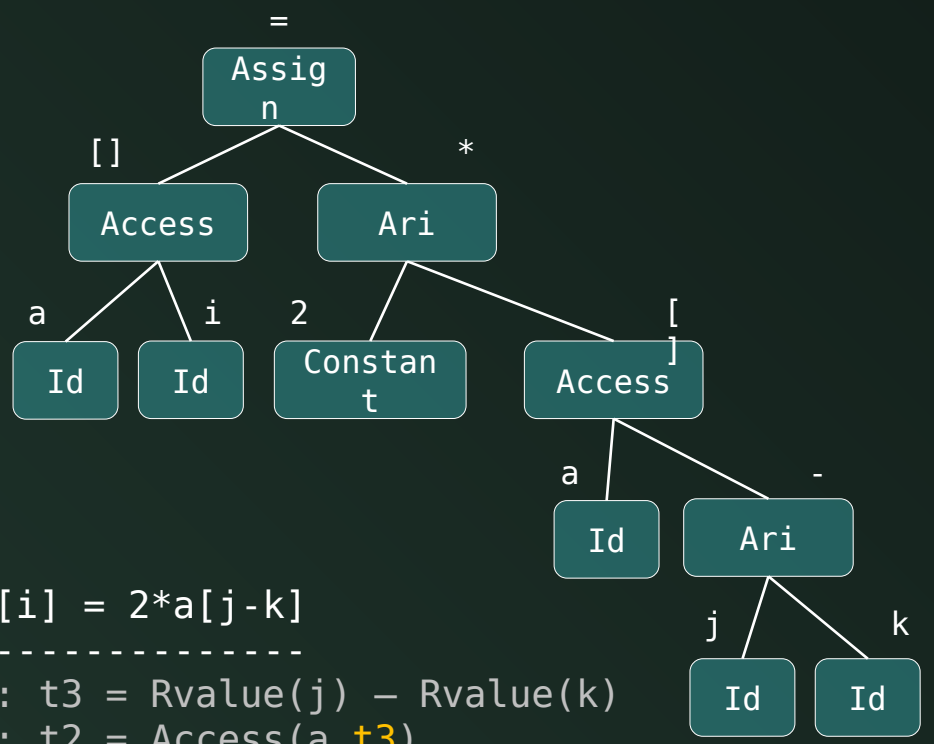



```
a[i] = 2*a[j-k]
---------------
1: t3 = Rvalue(j) – Rvalue(k)
2: t2 = Access(a,Rvalue(-))
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

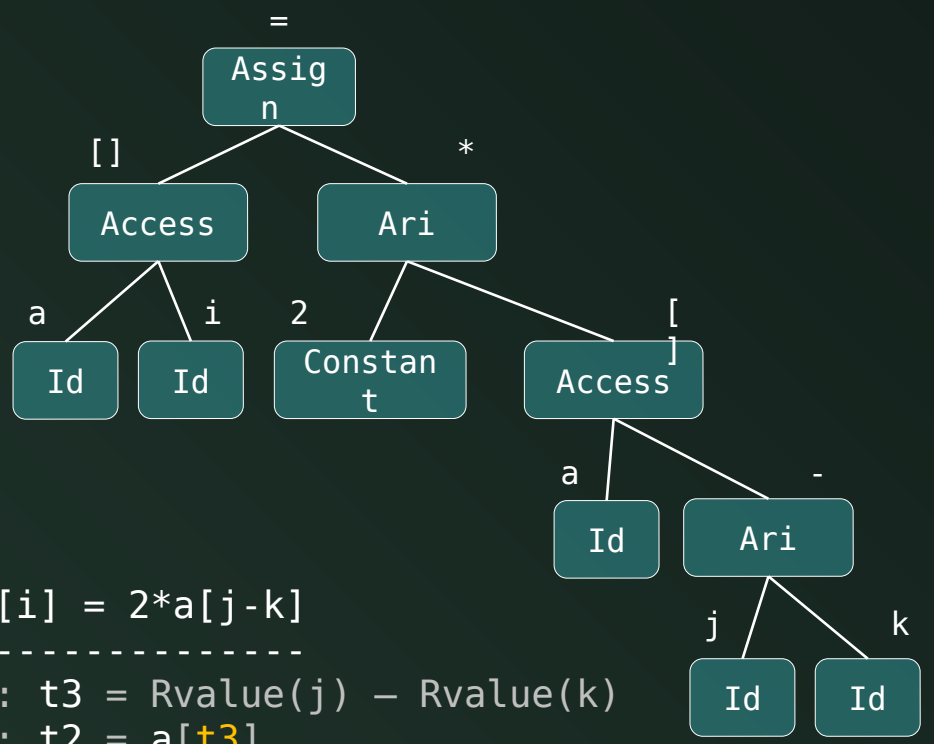



```
a[i] = 2*a[j-k]
---------------
1: t3 = Rvalue(j) – Rvalue(k)
2: t2 = Access(a,t3)
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

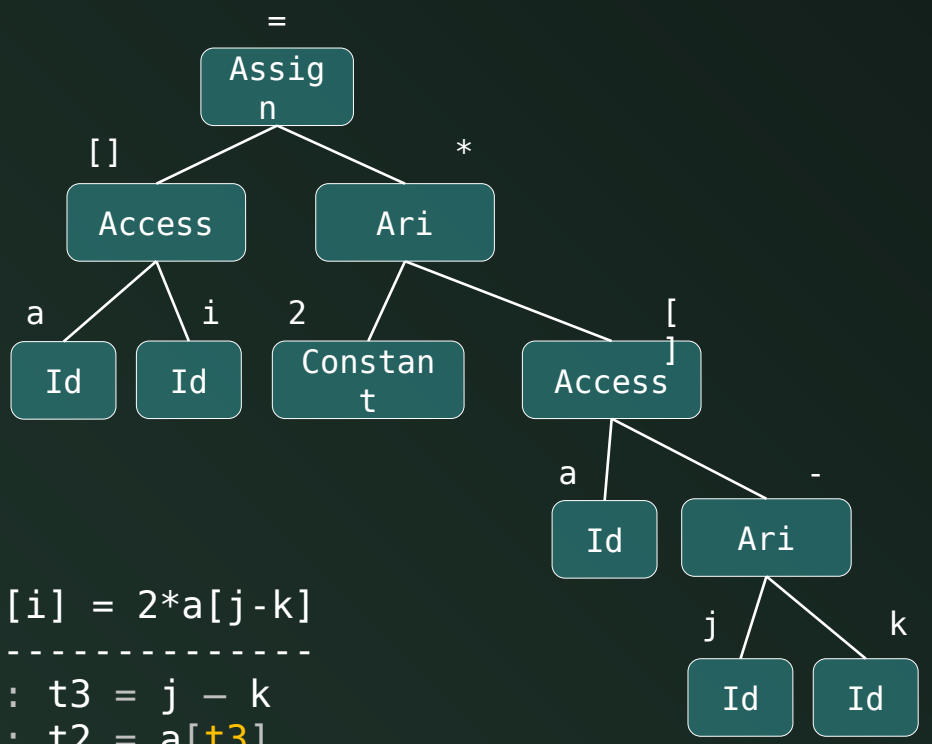



```
a[i] = 2*a[j-k]
---------------
1: t3 = Rvalue(j) – Rvalue(k)
2: t2 = a[t3]
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

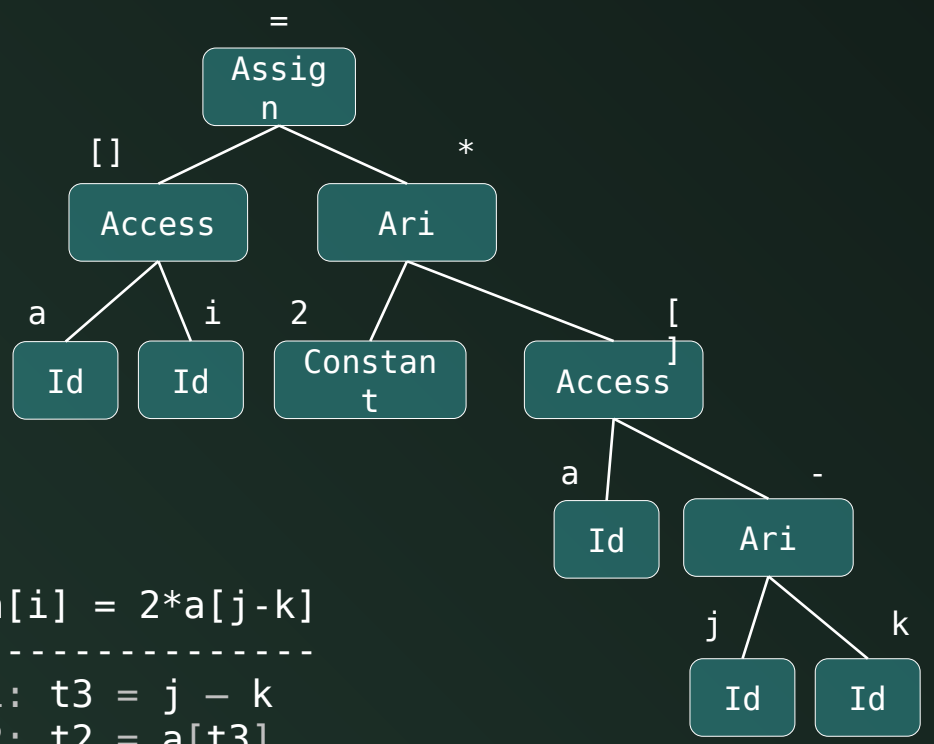



```
a[i] = 2*a[j-k]
---------------
1: t3 = j - k
2: t2 = a[t3]
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

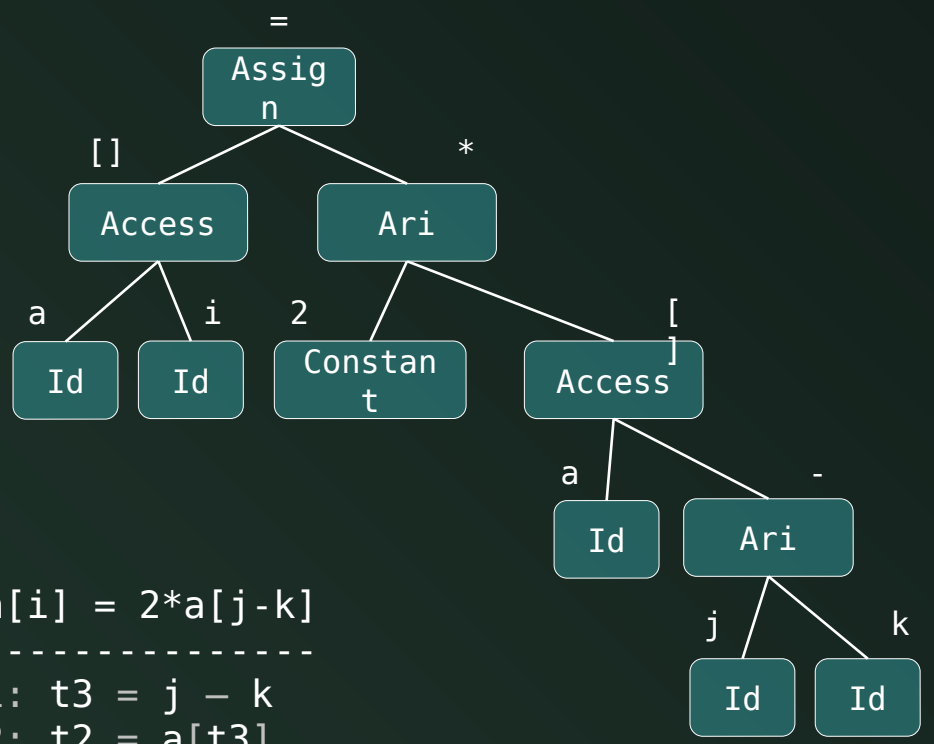



```
a[i] = 2*a[j-k]
---------------
1: t3 = j - k
2: t2 = a[t3]
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração

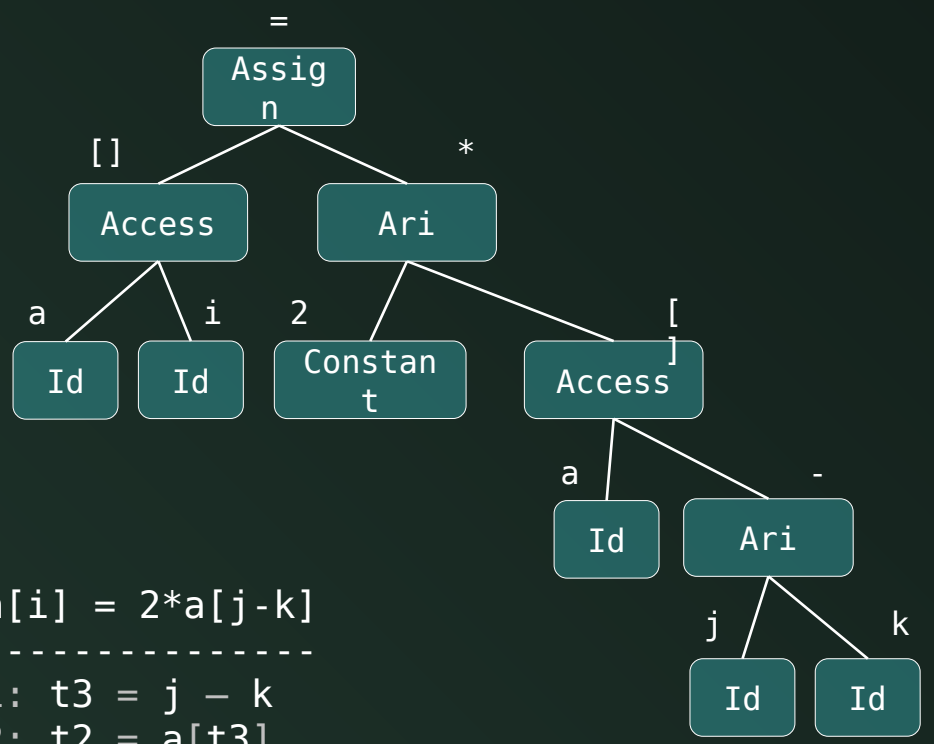



```
a[i] = 2*a[j-k]
---------------
1: t3 = j - k
2: t2 = a[t3]
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Exemplo de Geração




```
a[i] = 2*a[j-k]
---------------
1: t3 = j - k
2: t2 = a[t3]
3: t1 = 2 * t2
4: a[i] = t1
```

```
Expr Lvalue(n: Expr)
{
    if (n é um nó Id) return n;
    else if (n é um nó Access(y,z) e y é um nó Id)
        return new Access(y, Rvalue(z));
    else
       SyntaxError();
}

Expr Rvalue(n: Expr)
{
    if (n é um Id ou Constant) return n;
    else if (n é um nó Ari(op,y,z) ou um nó Rel(op,y,z)) {
        t = novo temporário;
        emite string para t = Rvalue(y) op Rvalue(z);
        return um novo nó para t;
    }
    else if (n é um nó Access(y,z)) {
        t = novo temporário;
        chame Lvalue(n), que retorna Access(y,z');
        emite string para t = Access(y,z');
        return um novo nó para t;
    }
}
```

Resumo

- A construção de representações auxiliares para o código fonte permite a realização de tarefas de análise e síntese:
  - Árvores de sintaxe
    - Permitem a execução de verificações estáticas sobre o programa:
      - Verificações sintáticas
      - Verificações de tipo
  - Códigos de três endereços
    - Criados a partir das árvores de sintaxe
      - Fornecem uma representação de mais baixo nível do programa
      - Permitem a otimização antes da geração de código objeto