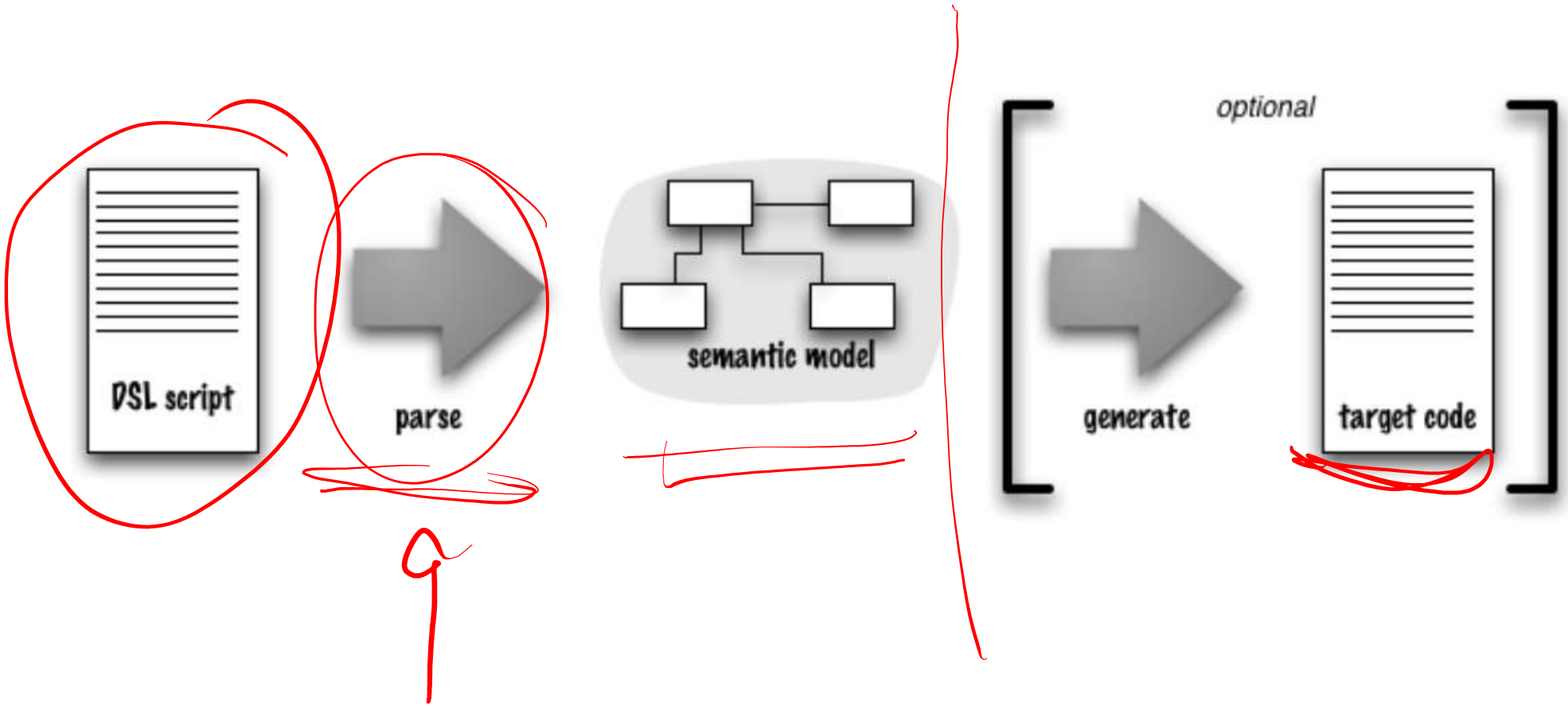
Linguagens de Domínio Específico

Fabio Mascarenhas – 2016.1

http://www.dcc.ufrj.br/~fabiom/dsl

Processamento de uma DSL



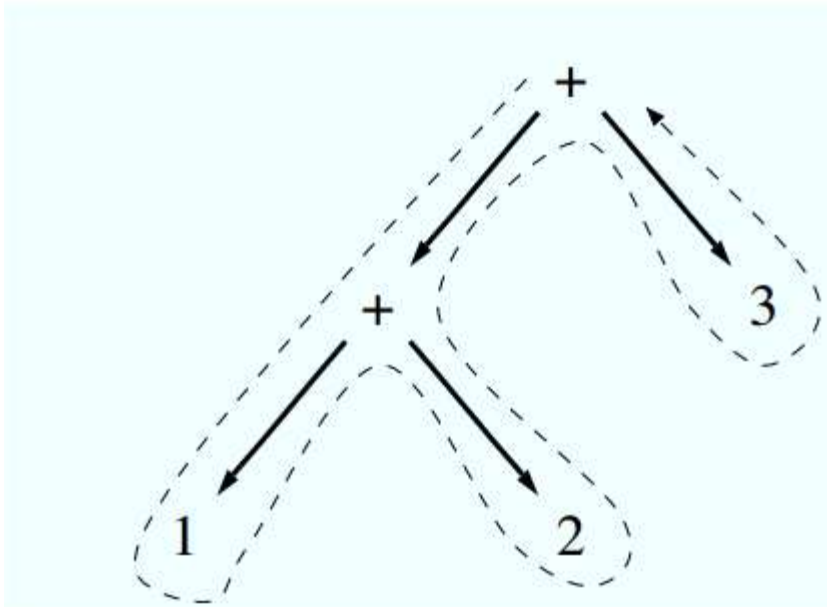
Percorrendo ASTs (1)

- Depois de construir a AST precisamos percorrê-la, provavelmente mais de uma vez:
  - Associar usos de nomes no programa a suas declarações
  - Verificar se o programa tem erros no uso de suas operações
  - Transformar a AST em um *modelo semântico* capaz de ser executado e/ou transformado em código final
- Vamos ver duas técnicas para escrever esses percursos da AST

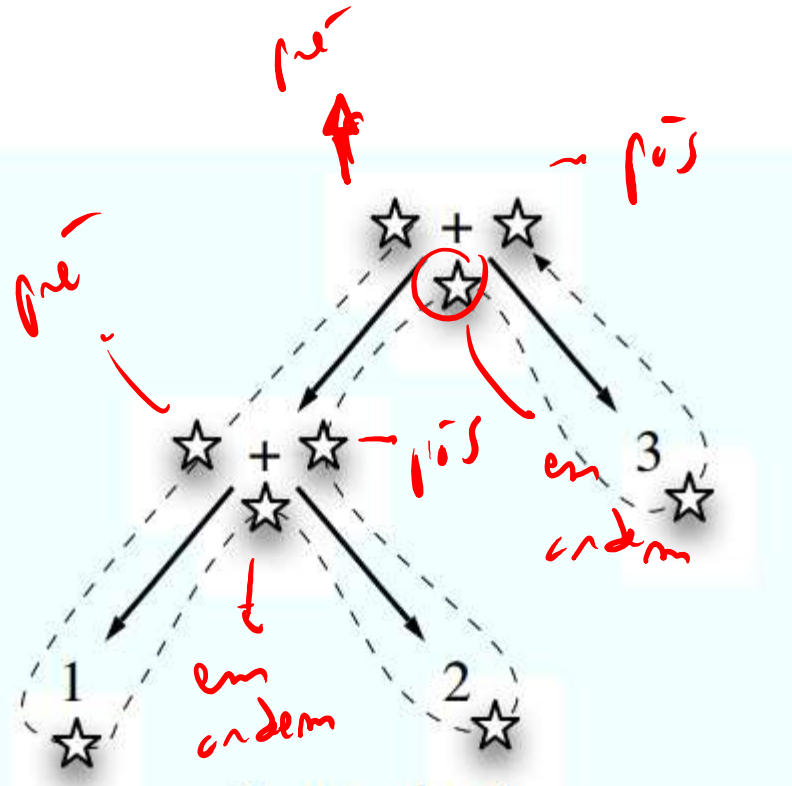
Percorrendo ASTs (2)

- Em um percurso de AST visitamos cada nó da árvore e executamos alguma ação
- Todos os percursos que vamos fazer vão ser percursos *em profundidade*: durante a visita a um nó visitamos cada um de seus filhos (e por sua vez os filhos deles serão visitados recursivamente)
- O que varia é a natureza das ações executadas, e quando no processo da visita elas são executadas
- Quase todo percurso que faremos também *modifica* a árvore enquanto ela é percorrida

Percurso em profundidade e ações



Depth first tree walk path:
node discovery, finishing



Traversal order:
action execution opportunities

Percurso embutido na AST

- Uma maneira de implementar um percurso é como um método polimórfico implementado em cada classe da AST

```
public class IntNode extends ExprNode {
    public void walk() { ; } // no children; nothing to do
}

public class AddNode extends ExprNode {
    ExprNode left, right; // named, node-specific, irregular children
    public void walk() {
        left.walk();      // walk the left operand subtree
        right.walk();     // walk the right operand subtree
    }
}
```

```
public void walk() {
    «preorder-action»
    left.walk();
    «inorder-action»
    right.walk();
    «postorder-action»
}
```

Percurso embutido – vantagens e desvantagens

- Essa técnica de implementação tem a vantagem da simplicidade
- Outra vantagem é poder manter o contexto de cada percurso nos parâmetros do método
- O grande problema é que a implementação do percurso fica espalhada entre todas as classes da AST, o que torna difícil ter uma visão geral da mesma
- Pode ser difícil entender o funcionamento do percurso tomando cada pedaço isoladamente

Padrão *Visitor*

- A outra técnica para implementar um percurso consiste em usar o padrão *Visitor*

- A ideia é encapsular todo o percurso como uma implementação de um visitor para aquela AST, com cada tipo de nó sendo um método do visitor

- Os nós simplesmente implementam um método `visit(Visitor v)` que chama o método em v correspondente àquele nó, passando o próprio nó

```
/** A generic heterogeneous tree node used in our vector math trees */
public abstract class VecMathNode extends HeteroAST {
    public VecMathNode() {;}
    public VecMathNode(Token t) { this.token = t; }
    public abstract void visit(VecMathVisitor visitor); // dispatcher
}
```

Implementando o Visitor

- Note que o visitor usa sobrecarga, mas poderia usar nomes diferentes para cada método visit se a linguagem de implementação não tiver sobrecarga

```
public interface VecMathVisitor {
    void visit(AssignNode n);
    void visit(PrintNode n);
    void visit(StatListNode n);
    void visit(VarNode n);
    void visit(AddNode n);
    void visit(DotProductNode n);
    void visit(IntNode n);
    void visit(MultNode n);
    void visit(VectorNode n);
}
```

```
public class PrintVisitor implements VecMathVisitor {
    public void visit(AssignNode n) {
        n.id.visit(this);
        System.out.print("=");
        n.value.visit(this);
        System.out.println();
    }
```

Contexto e vantagens

- Para poder passar algum parâmetro de contexto para o visitor, assim como retornar um valor, podemos usar uma interface genérica
- Podemos até ter mais de uma, para visitors que não precisam de parâmetros, visitors com um parâmetro, visitors com dois parâmetros etc., à custa de precisar de mais métodos `visit` nas classes da AST
- Também podemos guardar informações de contexto na própria instância do visitor
- Apesar de mais complexidade e mais ruído nas chamadas recursivas, implementar um percurso com um visitor deixa o percurso mais fácil de escrever e manter