Conférence AST

ACU 2022 Team

Copyright

AST

Abstract Syntax Tree

$2 + 6 * 3$

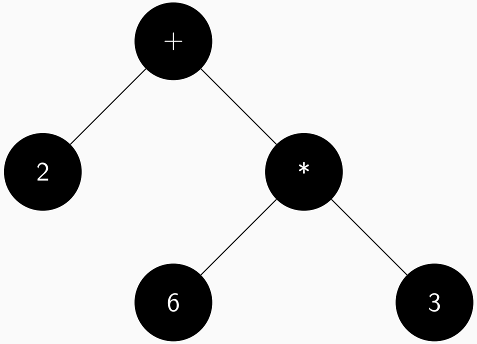


Figure 1: A simple example

Evaluate the AST - Data representation

```c
enum node_type
{
    ADD,
    MULT,
    NUMBER
};
struct node
{
    enum node_type type;
    struct node *left;
    struct node *right;
    int value;
};
```

Evaluate the AST

```c
int evaluate( struct node *ast_node)
{
    if (ast_node->type == ADD)
    {
        return evaluate(ast_node->left) + evaluate(ast_node->right);
    }
    else if (ast_node->type == MULT)
    {
        return evaluate(ast_node->left) * evaluate(ast_node->right);
    }
    else
    {
        return ast_node->value;
    }
}
```

Evaluate the AST - Example

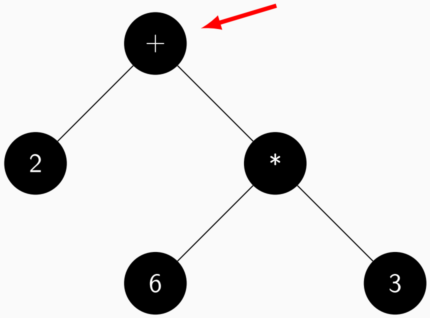


Figure 2: AST traversal example

Return statement

```
    return evaluate(node->left) + evaluate(node->right);
```

Evaluate the AST - Example

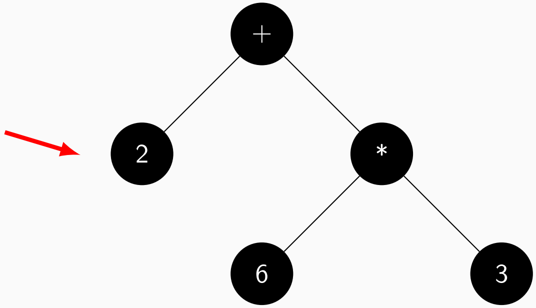


Figure 3: AST traversal example

Return statement

```c
return node->value;
// 2
```

Evaluate the AST - Example

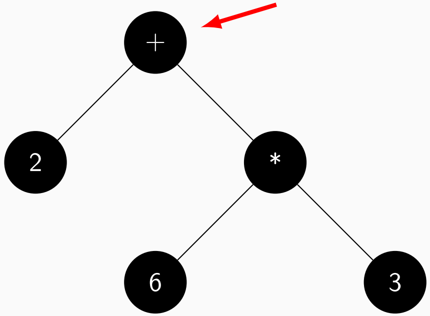


Figure 4: AST traversal example

Return statement

```
return evaluate(node->left) + evaluate(node->right);
// 2
```

Evaluate the AST - Example

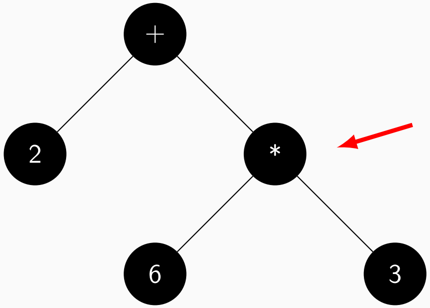


Figure 5: AST traversal example

Return statement

```
    return evaluate(node->left) * evaluate(node->right);
    // 2
```

Evaluate the AST - Example

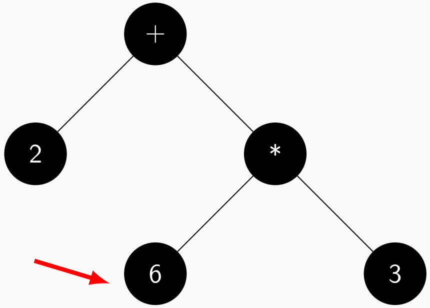


Figure 6: AST traversal example

Return statement

```
return node->value;
// 2
// 6
```

Evaluate the AST - Example

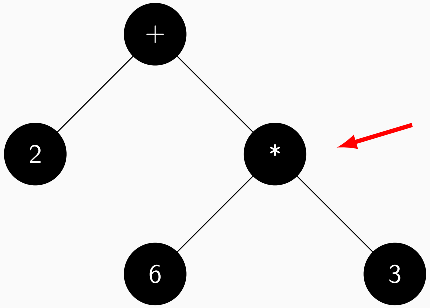


Figure 7: AST traversal example

Return statement

```
return evaluate(node->left) * evaluate(node->right);
// 2
// 6
```

Evaluate the AST - Example

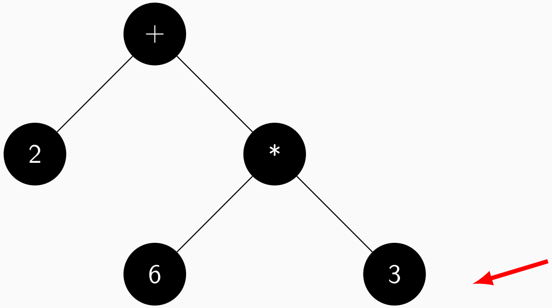


Figure 8: AST traversal example

Return statement

```
return node->value;
// 2
// 6 3
```

Evaluate the AST - Example

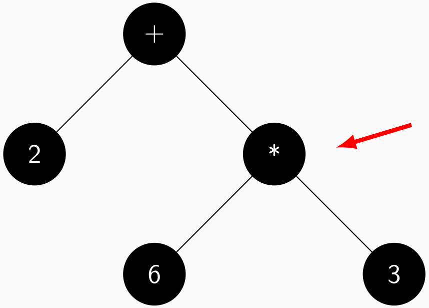


Figure 9: AST traversal example

Return statement

```
return evaluate(node->left) * evaluate(node->right);
// 2
// 6 * 3
```

Evaluate the AST - Example

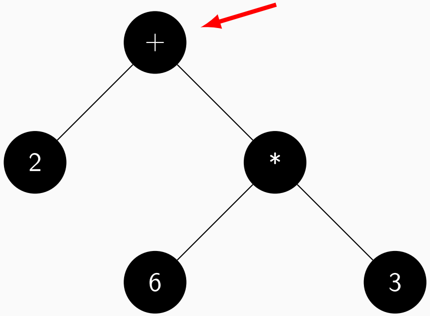


Figure 10: AST traversal example

Return statement

```
return evaluate(node->left) + evaluate(node->right);
// 2 + 6 * 3
```

Evaluate the AST - MyFind case

```c
int evaluate( struct node *ast_node)
{
    if (ast_node->type == OR)
        return evaluate(ast_node->left) || evaluate(ast_node->right);
    else if (ast_node->type == AND)
        return evaluate(ast_node->left) && evaluate(ast_node->right);
    else
        for ( int i = 0; i < funs_len; i++)
            if (funs[i].type == ast_node->type)
                return funs[i].fun(ast_node->value);
}
```

Evaluate the AST - MyFind case

```c
struct function
{
    enum node_type type;
    int (*fun)( char *);
};

struct function funs[];

for ( int i = 0; i < funs_len; i++)
    if (funs[i].type == node->type)
        return funs[i].fun();
```

Build the AST

FROM

TO

```
char *input = "2 + 6 * 3";
```

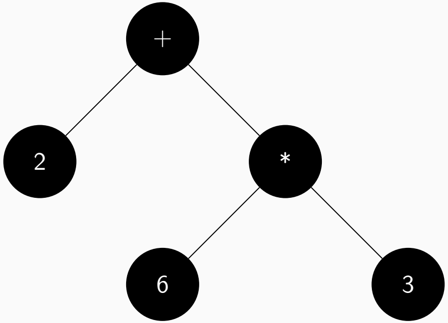


Figure 11: From string to AST nodes

Build the AST

```c
enum node_type
{
    ADD,
    MULT,
    NUMBER
};

struct node
{
    enum node_type type;
    int value;
};
```

Build the AST - From string to tokens

FROM

```c
char *input = "2 + 6 * 3";
```

TO

```c
struct token tokens[] = {
    {NUMBER, 2},
    {ADD, 0},
    {NUMBER, 6},
    {MULT, 0},
    {NUMBER, 3}
};
```

Build the AST - From tokens to nodes

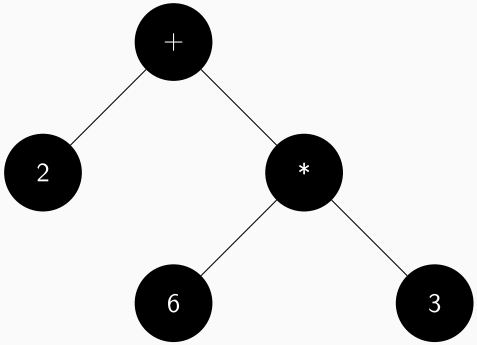


Figure 12: To AST nodes

Build the AST - From tokens to nodes

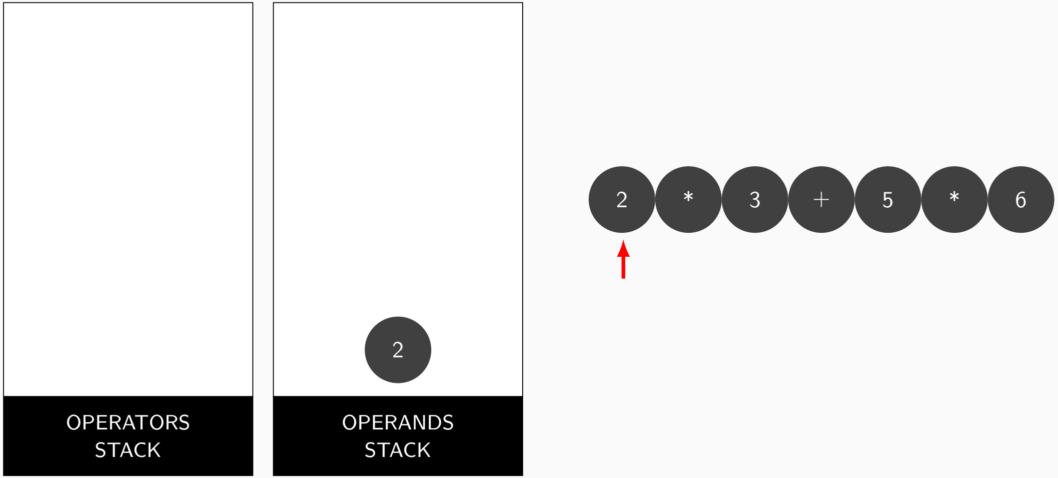


Figure 13: From tokens to nodes

Build the AST - From tokens to nodes

Code translation

```
{
    NUMBER, // Node type
    NULL, // Left node
    NULL, // Right node
    2 // Value
}
```

Build the AST - From tokens to nodes

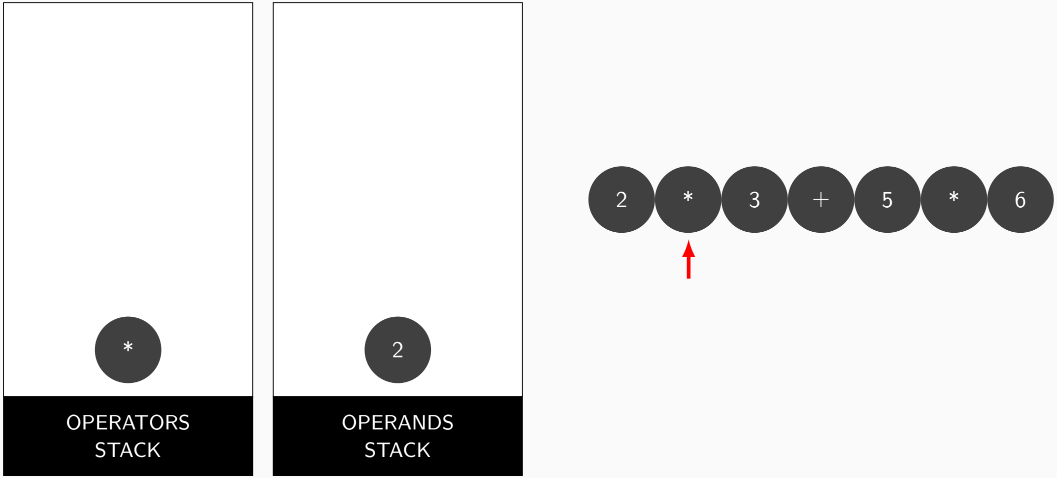


Figure 14: From tokens to nodes

Build the AST - From tokens to nodes

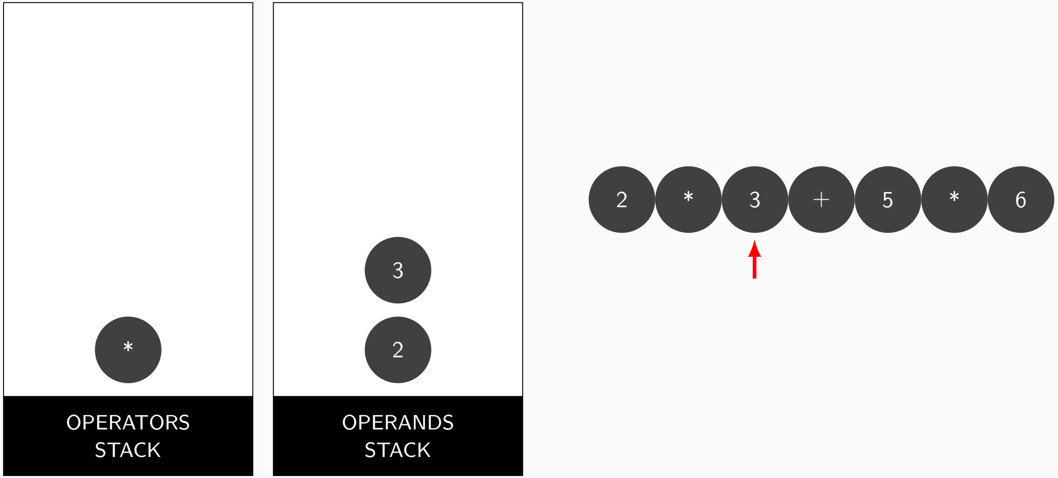


Figure 15: From tokens to nodes

Build the AST - From tokens to nodes

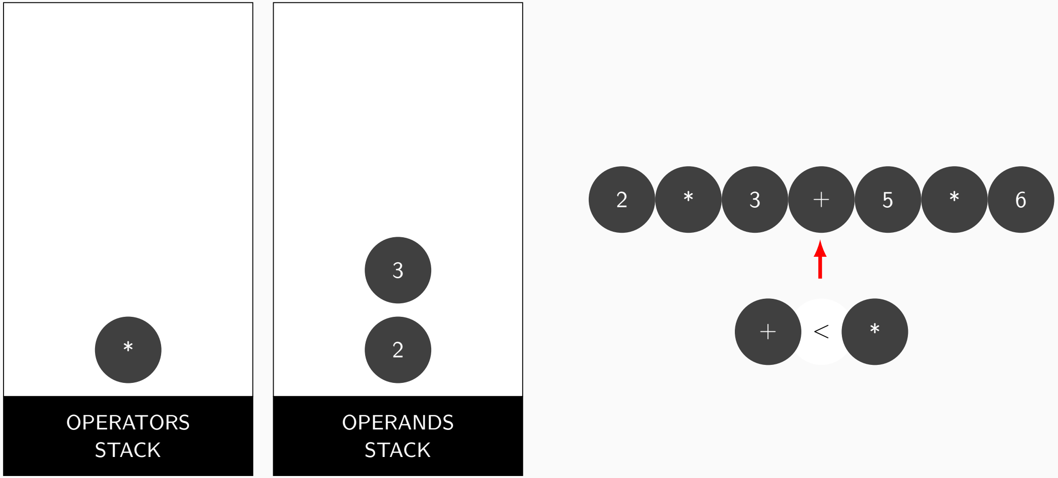


Figure 16: From tokens to nodes

Build the AST - From tokens to nodes

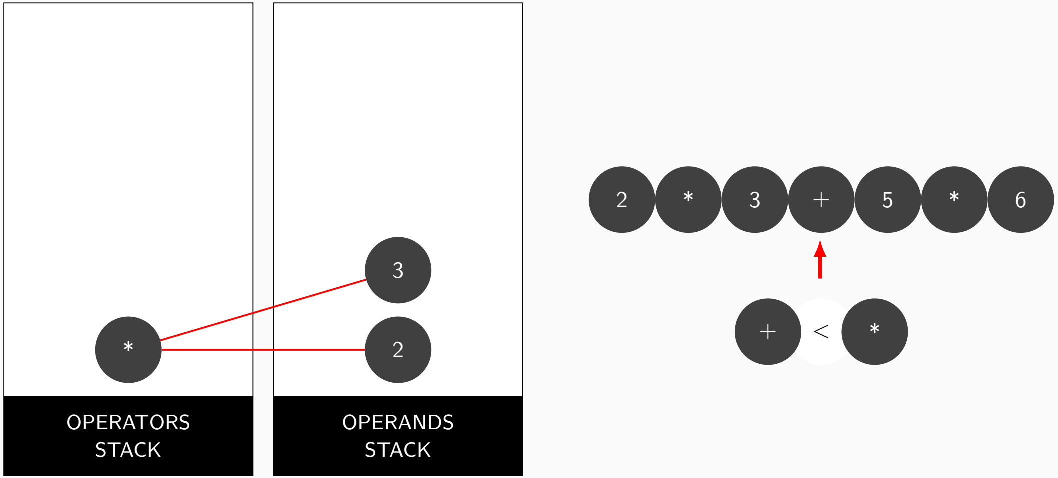


Figure 17: From tokens to nodes

Build the AST - From tokens to nodes

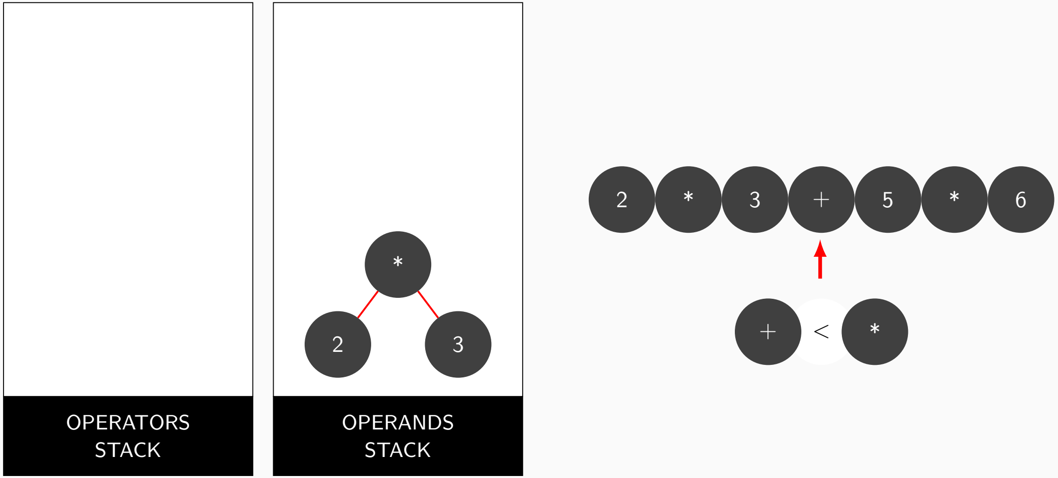


Figure 18: From tokens to nodes

Build the AST - From tokens to nodes

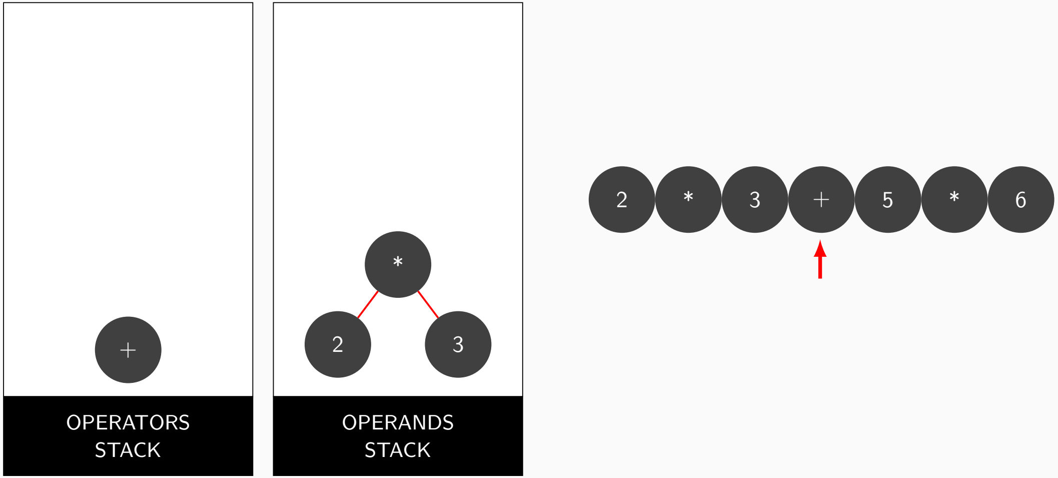


Figure 19: From tokens to nodes

Build the AST - From tokens to nodes

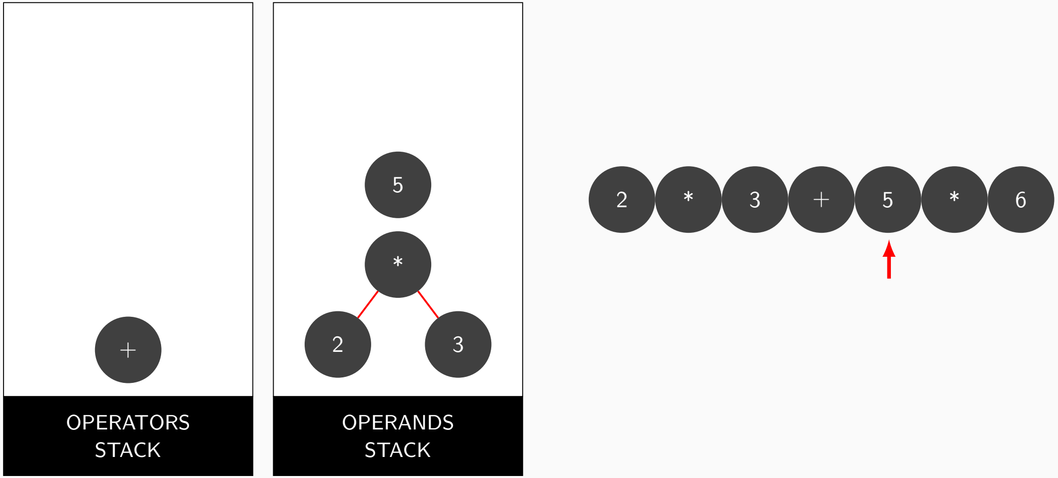


Figure 20: From tokens to nodes

Build the AST - From tokens to nodes

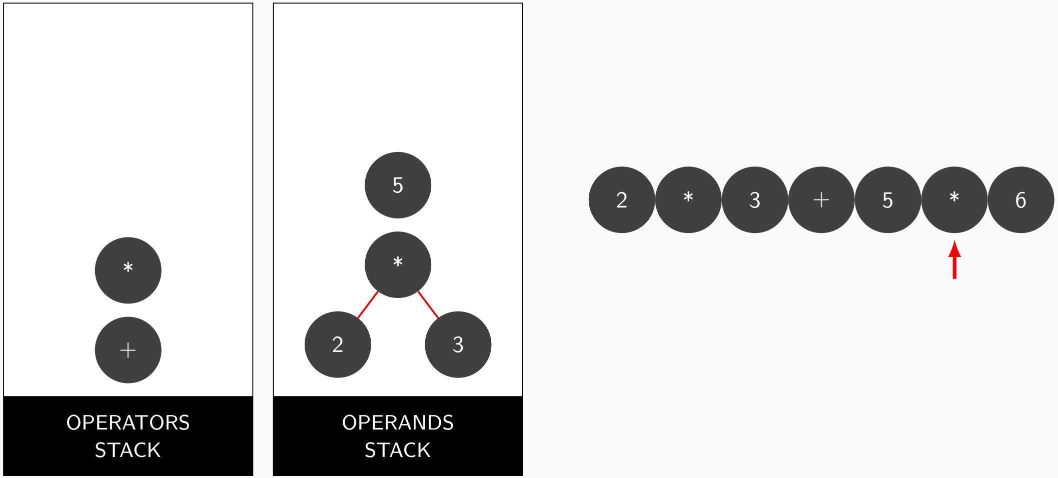


Figure 21: From tokens to nodes

Build the AST - From tokens to nodes

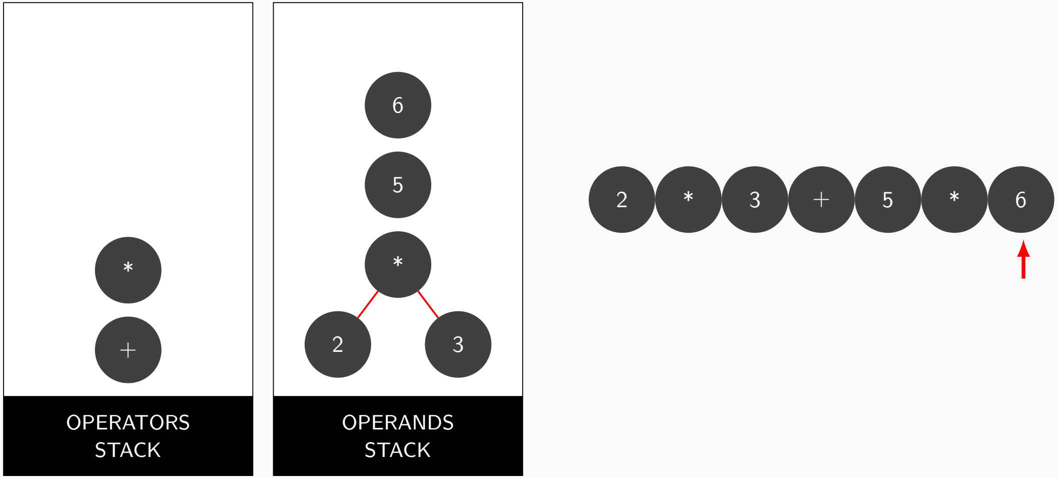


Figure 22: From tokens to nodes

Build the AST - From tokens to nodes

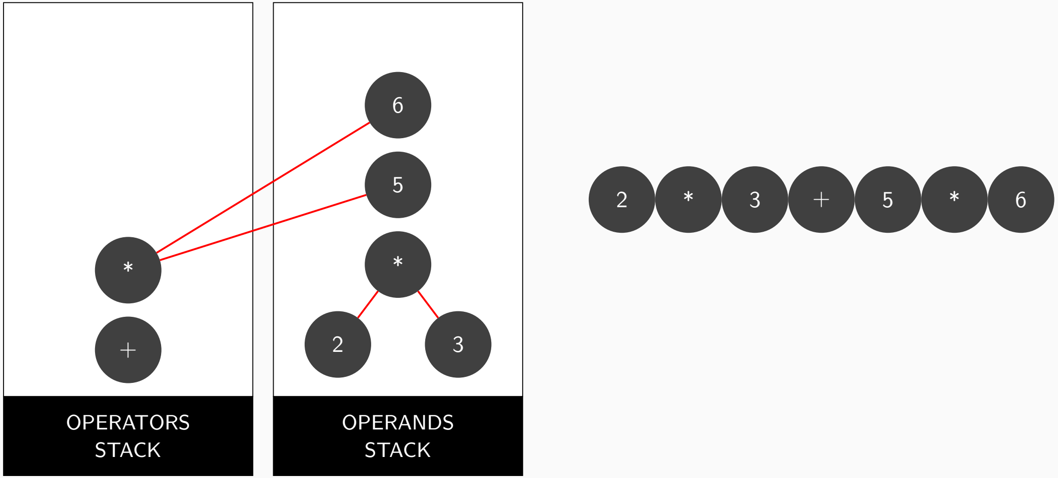


Figure 23: From tokens to nodes

Build the AST - From tokens to nodes

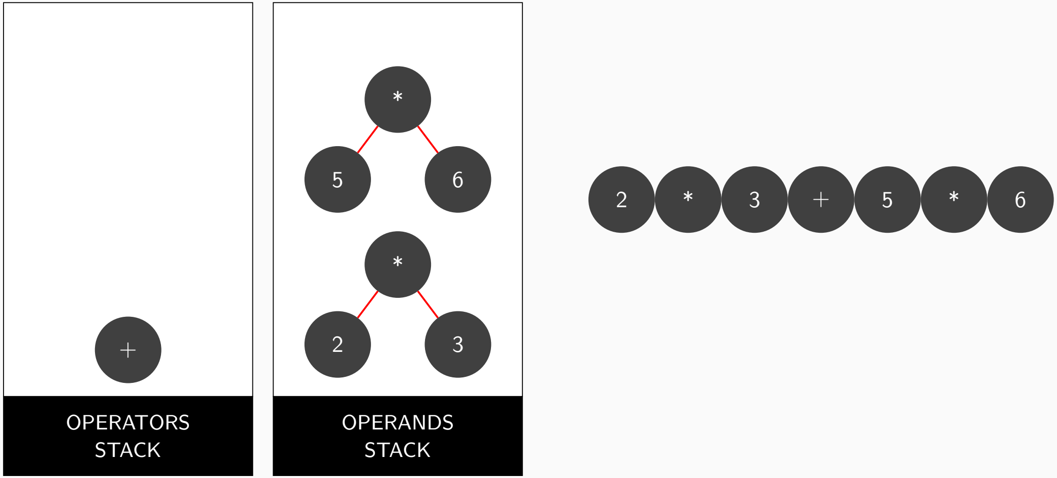


Figure 24: From tokens to nodes

Build the AST - From tokens to nodes

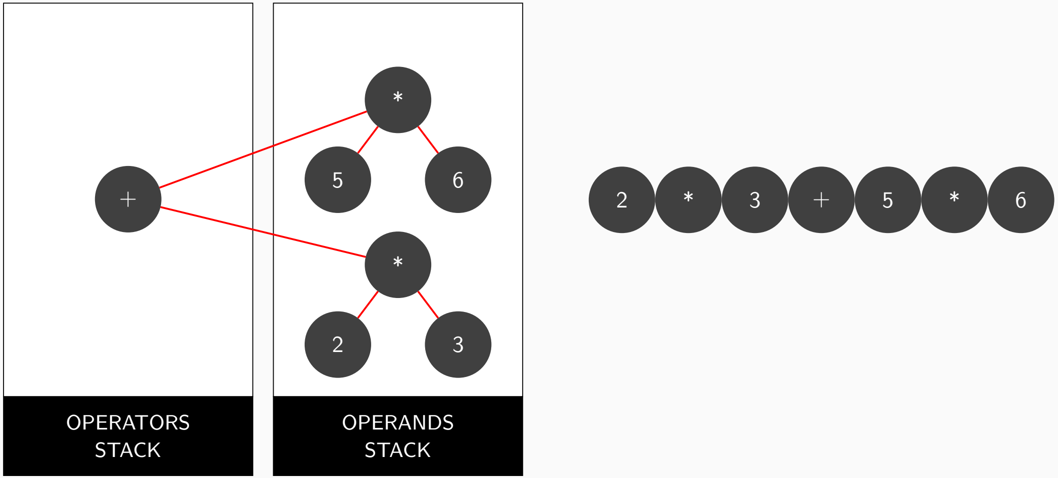


Figure 25: From tokens to nodes

Build the AST - From tokens to nodes

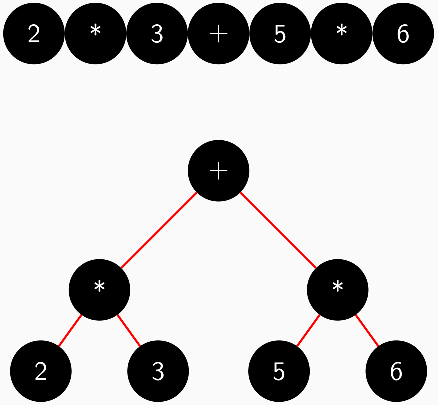
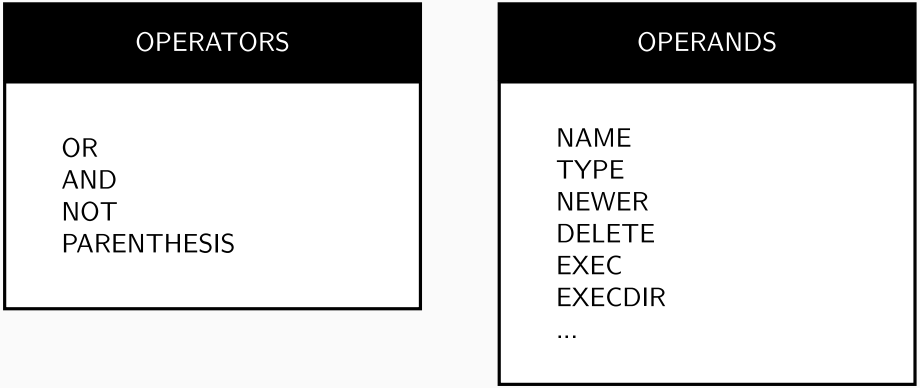


Figure 26: From tokens to nodes

Build the AST - MyFind case



OPERATORS	OPERANDS
OR	NAME
AND	TYPE
NOT	NEWER
PARENTHESIS	DELETE
	EXEC
	EXECDIR
	...

Figure 27: MyFind tokens

Build the AST - MyFind case

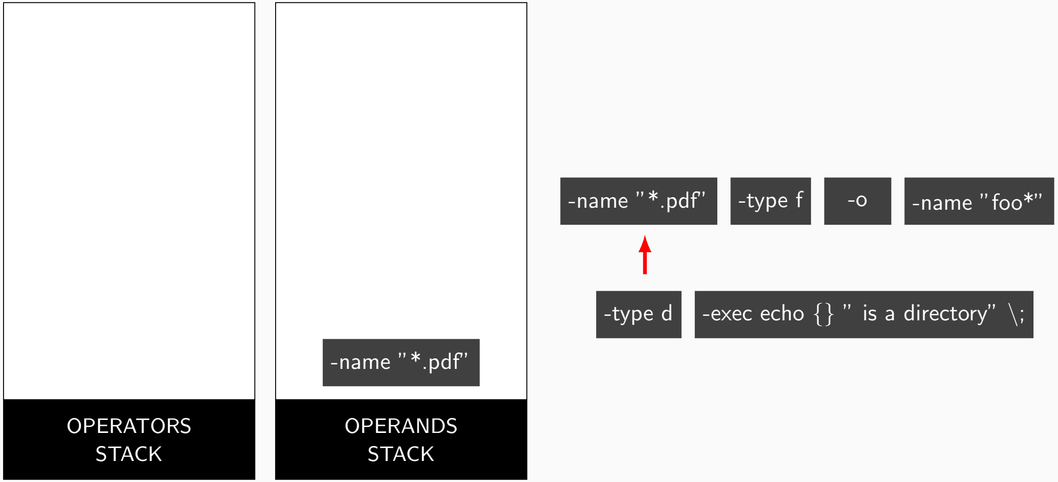


Figure 28: MyFind tree construction

Build the AST - MyFind case

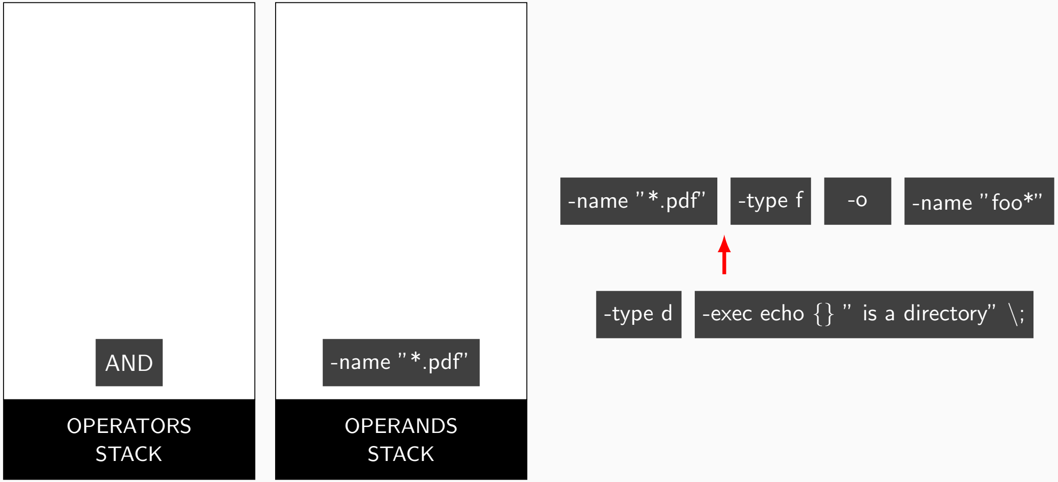


Figure 29: MyFind tree construction

Build the AST - MyFind case

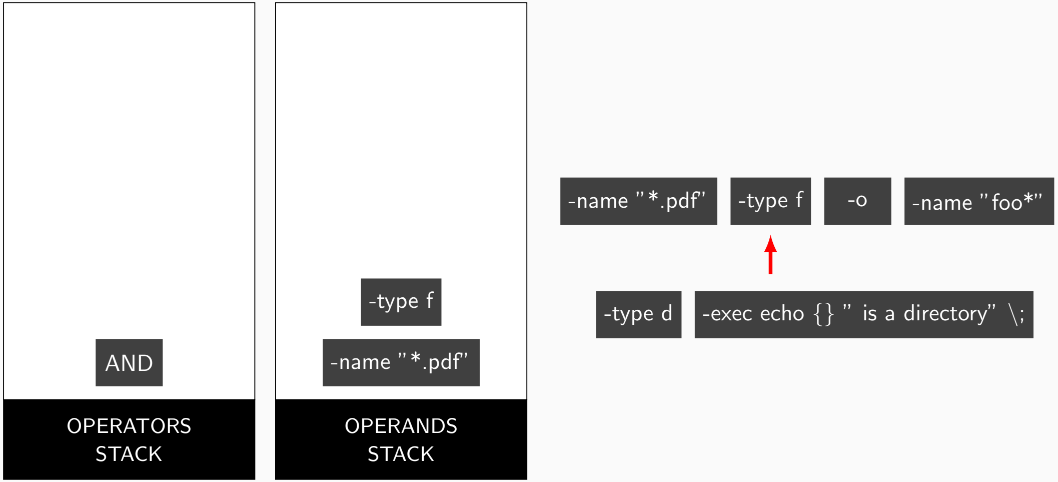


Figure 30: MyFind tree construction

Build the AST - MyFind case

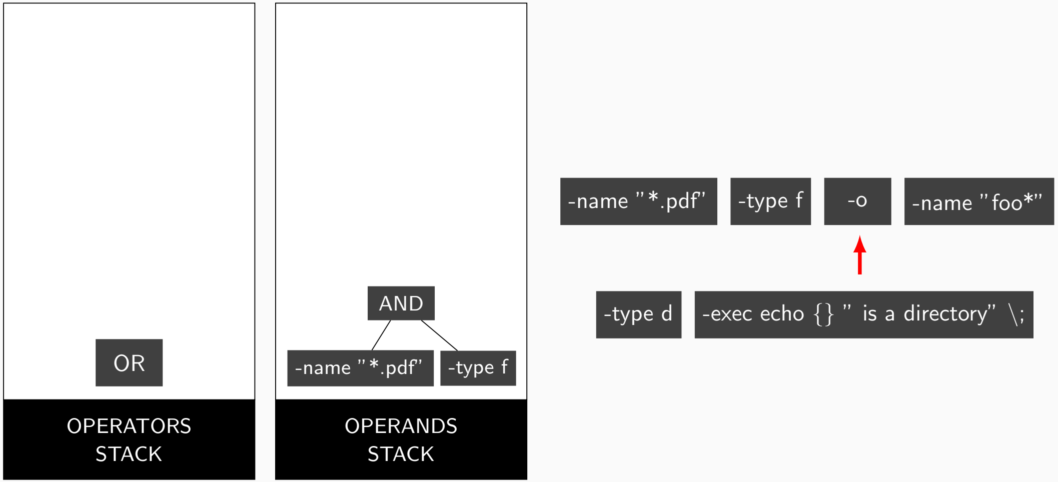


Figure 31: MyFind tree construction

Build the AST - MyFind case

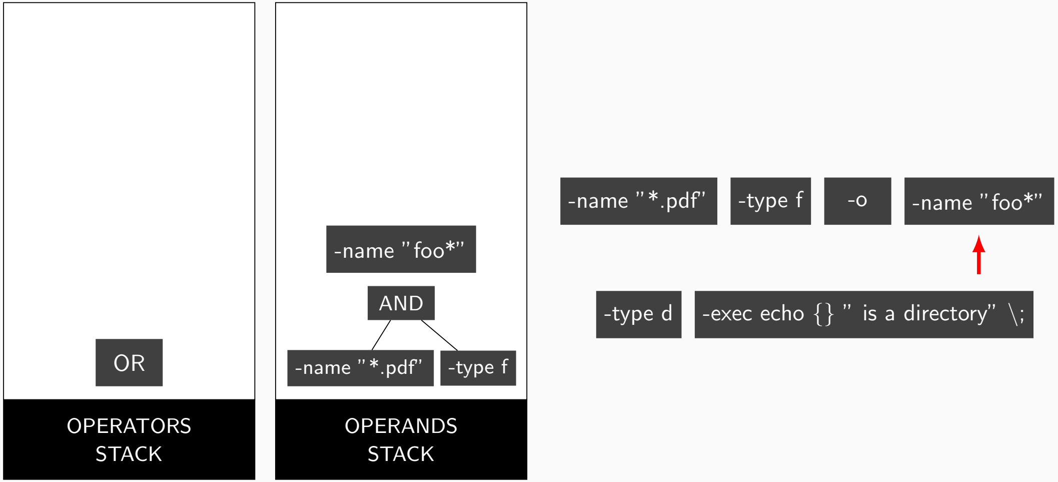


Figure 32: MyFind tree construction

Build the AST - MyFind case

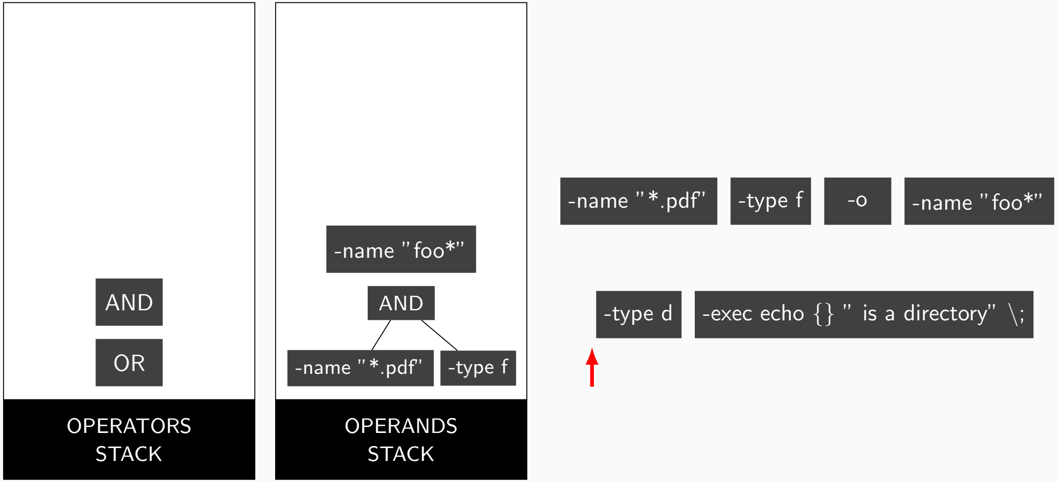


Figure 33: MyFind tree construction

Build the AST - MyFind case

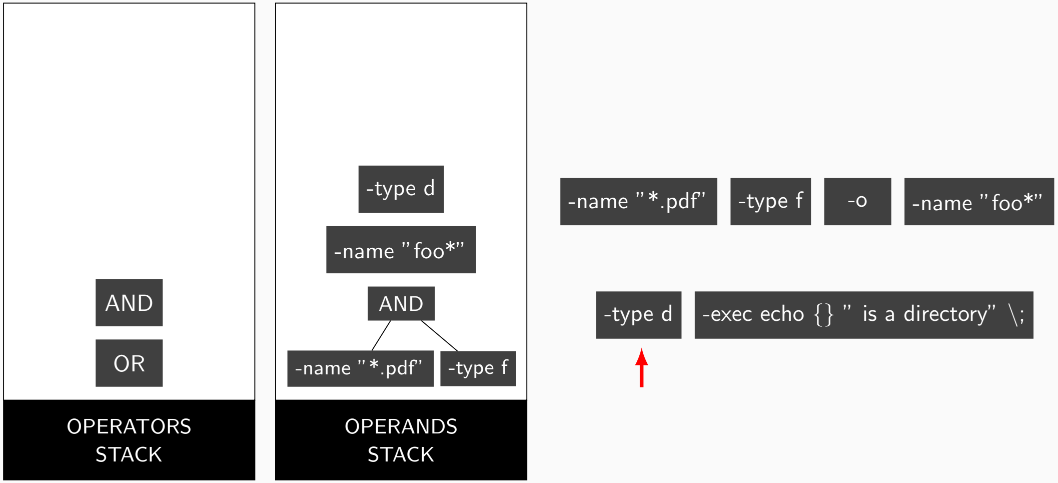


Figure 34: MyFind tree construction

Build the AST - MyFind case

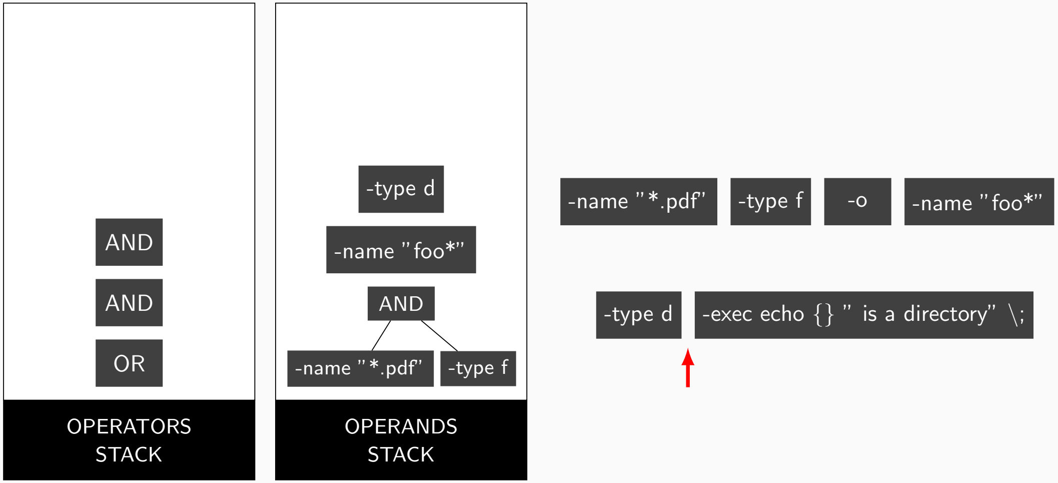


Figure 35: MyFind tree construction

Build the AST - MyFind case

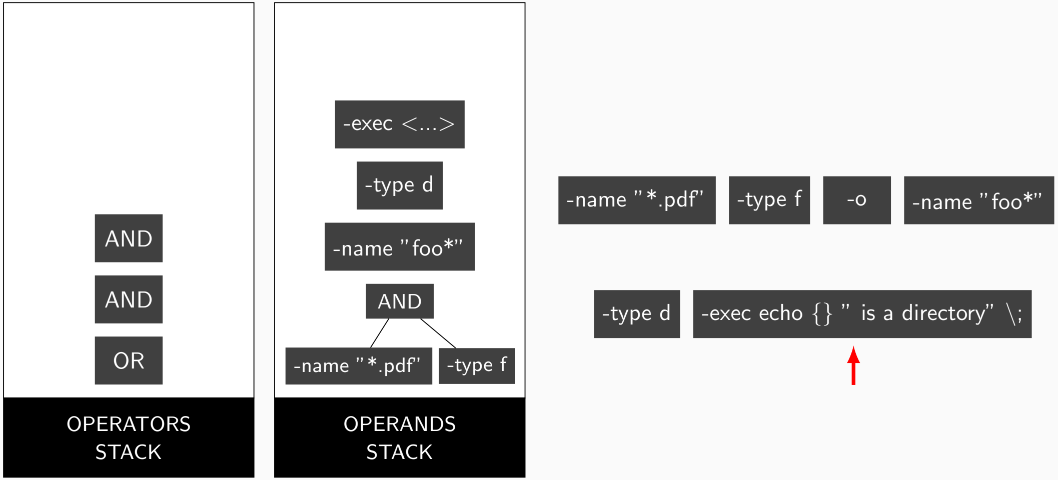


Figure 36: MyFind tree construction

Build the AST - MyFind case

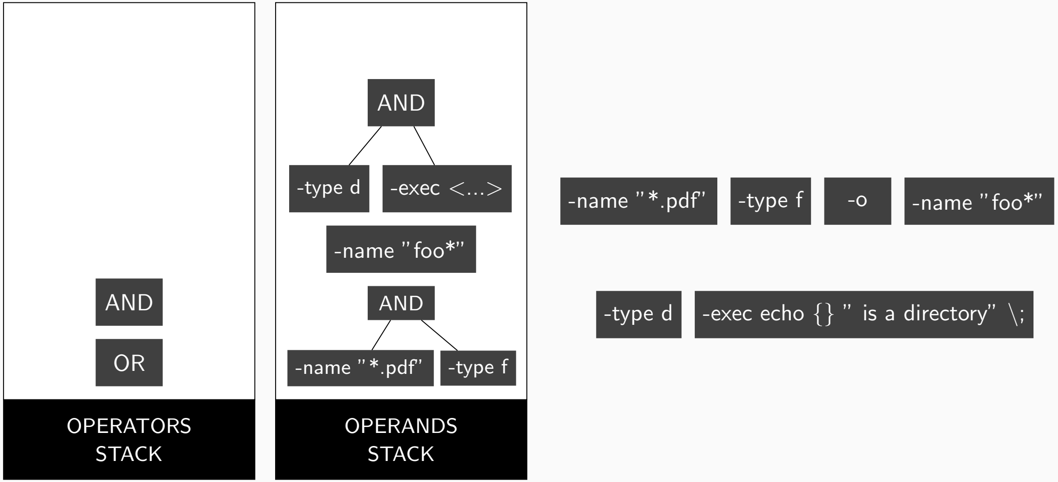


Figure 37: MyFind tree construction

Build the AST - MyFind case

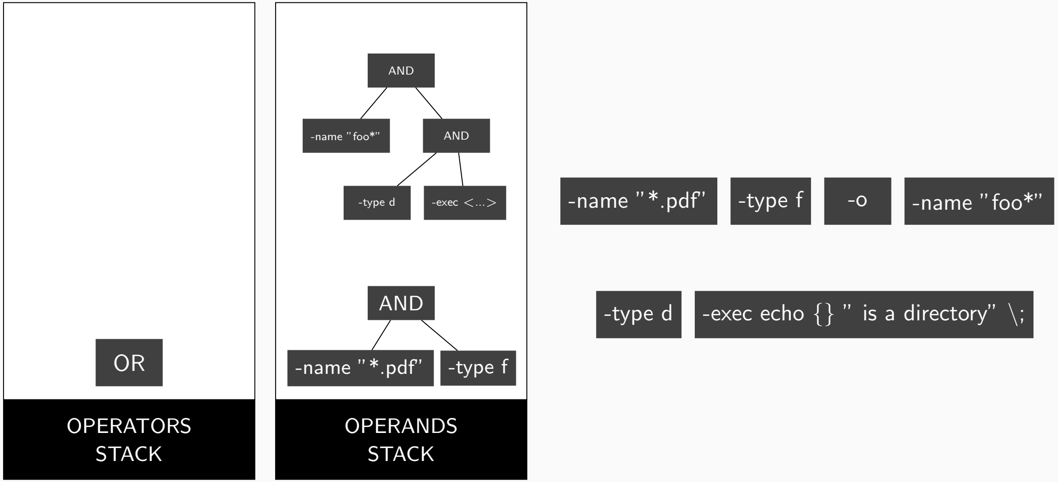


Figure 38: MyFind tree construction

Build the AST - MyFind case

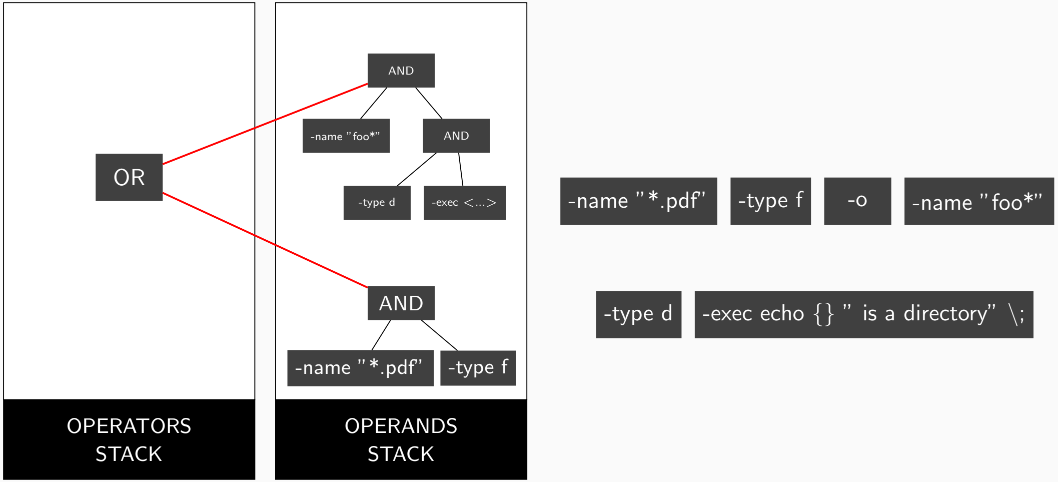


Figure 39: MyFind tree construction

Build the AST - MyFind case

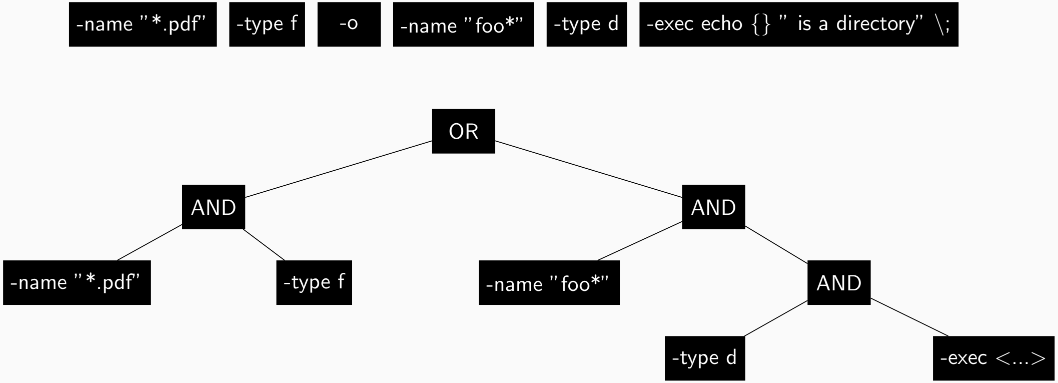


Figure 40: MyFind tree construction