

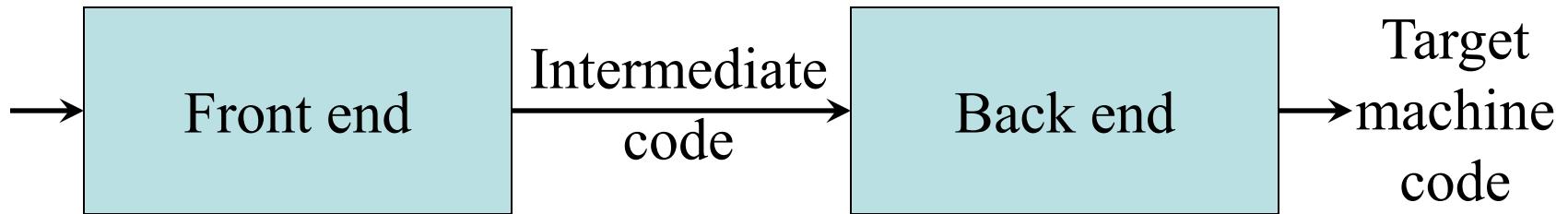
Intermediate Code Generation

Chapter 6

Course Instructor: Dr. Hamra Afzaal

Intermediate Code Generation

- Facilitates *retargeting*: enables attaching a back end for the new machine to an existing front end



- Enables machine-independent code optimization

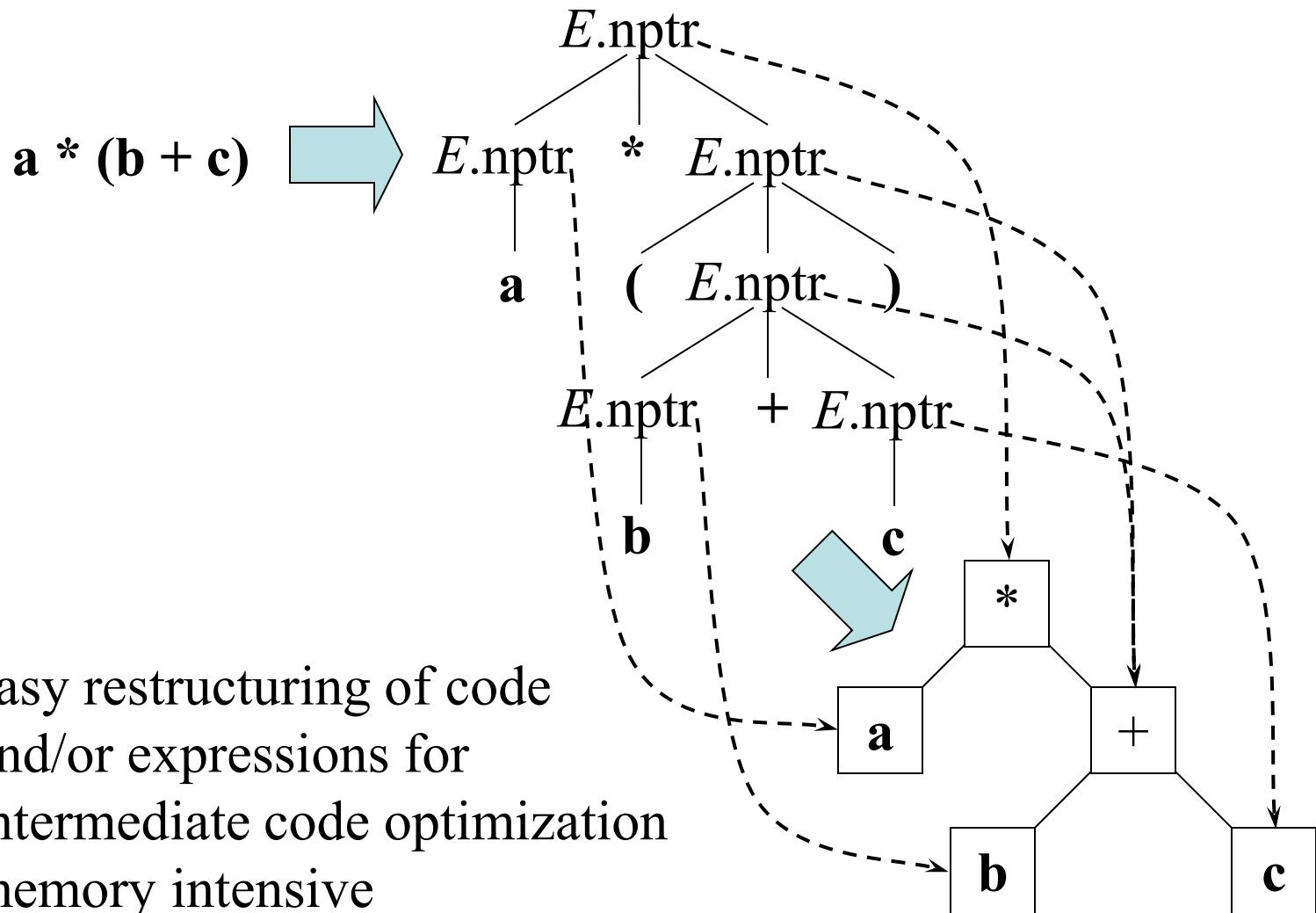
Intermediate Representations

- *Graphical representations* (e.g. AST)
- *Postfix notation*: operations on values stored on operand stack
- *Three-address code*: (e.g. *triples*)
 $x := y \text{ op } z$
- *Two-address code*:
 $x := \text{op } y$
which is the same as $x := x \text{ op } y$

Syntax-Directed Translation of Abstract Syntax Trees

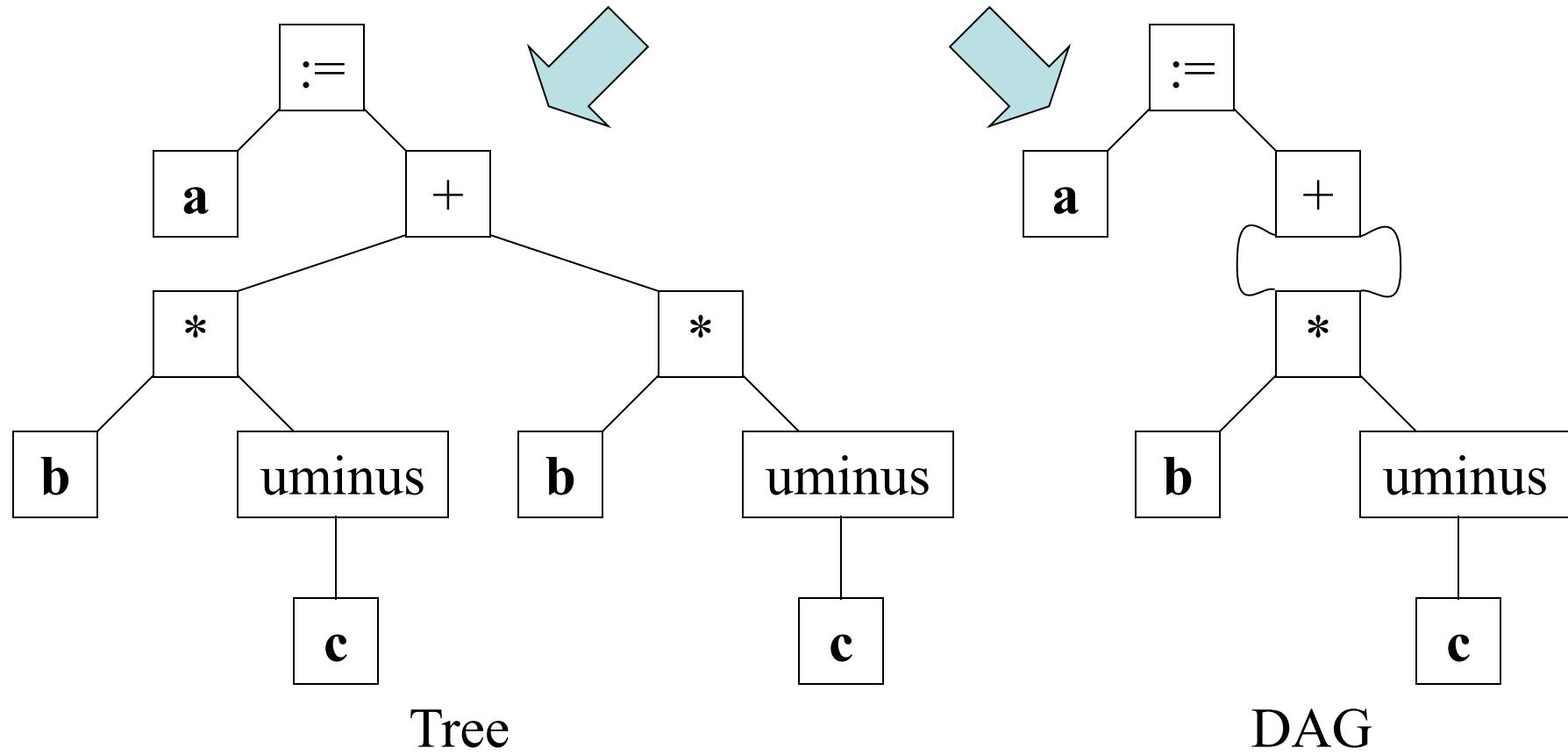
Production	Semantic Rule
$S \rightarrow \mathbf{id} := E$	$S.\text{nptr} := \text{mknode}(\text{`:='}, \text{mkleaf}(\mathbf{id}, \mathbf{id}.\text{entry}), E.\text{nptr})$
$E \rightarrow E_1 + E_2$	$E.\text{nptr} := \text{mknode}(\text{`+'}, E_1.\text{nptr}, E_2.\text{nptr})$
$E \rightarrow E_1 * E_2$	$E.\text{nptr} := \text{mknode}(\text{`*'}, E_1.\text{nptr}, E_2.\text{nptr})$
$E \rightarrow -E_1$	$E.\text{nptr} := \text{mknode}(\text{'uminus'}, E_1.\text{nptr})$
$E \rightarrow (E_1)$	$E.\text{nptr} := E_1.\text{nptr}$
$E \rightarrow \mathbf{id}$	$E.\text{nptr} := \text{mkleaf}(\mathbf{id}, \mathbf{id}.\text{entry})$

Abstract Syntax Trees



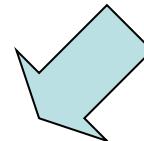
Abstract Syntax Trees versus DAGs

$a := b * -c + b * -c$



Postfix Notation

a := b * -c + b * -c



a b c uminus * b c uminus * + assign

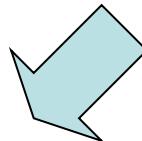
Postfix notation represents
operations on a stack

Pro: easy to generate

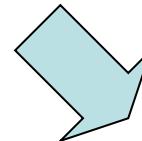
Cons: stack operations are more
difficult to optimize

Three-Address Code

a := b * -c + b * -c



```
t1 := - c
t2 := b * t1
t3 := - c
t4 := b * t3
t5 := t2 + t4
a   := t5
```



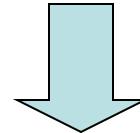
```
t1 := - c
t2 := b * t1
t5 := t2 + t2
a   := t5
```

Linearized representation
of a syntax tree

Linearized representation
of a syntax DAG

Example

```
i := 2 * n + k  
while i < 0 do  
    i := i - k
```



```
t1 := 2  
t2 := t1 * n  
t3 := t2 + k  
i := t3  
L1: if i = 0 goto L2  
    t4 := i - k  
    i := t4  
    goto L1  
L2:
```