**Translating Expressions to Stack Machine**

**Connection to Postfix**

**Example:** translate expression

(a\*b + b\*c + a\*c) \* 2

into posftix form.

Tree:

\*

+ 2

\* +

a b \* \*

b c a c

Postfix expression:

a b \* b c \* + a c \* + 2 \*

Translated expression:

iload slot\_a

iload slot\_b

imul

iload slot\_b

iload slot\_c

imul

iadd

iload slot\_a

iload slot\_c

imul

iadd

iconst\_2

imul

Compare to cubeArea in [Compiled Expression Examples](http://lara.epfl.ch/w/cc09:compiled_expression_examples)

In compiled code what corresponds to:

* ‘+’
* ‘\*’
* variables, like a,b,c
* constants, like 2

Conclusion: **Translating expressions to stack machine is like printing expression in postfix form.**

**Translation Rules**

For constant $c$,   
\begin{displaymath}
    [\![ c ]\!] = List(\textbf{bipush}(c))
\end{displaymath}  
The **iconst** is just special case of **bipush**

Translation for operations:   
\begin{displaymath}
    [\![ e_1 * e_2 ]\!] = [\![ e_1 ]\!] ::: [\![ e_2 ]\!] ::: List([\![*]\!])
\end{displaymath}  
and similarly for other operations.

* we will sometimes omit List(...) when it is clear from the context

Intuition: to evaluate e1 \* e2, an interpreter would

* evaluate e1
* evaluate e2
* perform operation on these two values

Compiled code should do the same

* generate code that will evaluate e1 and store it somewhere
* generate code that will evaluate e2 and store it somewhere else
* retrieve these two values and perform the operation on them

In stack machine, “somewhere” is an appropriate part of the stack

* we need to know that “somewhere” and “somewhere else” are different
* what would happen if compile code “ate” stack before and removed previous value?

Properties of generated code for expression:

* do not erase anything that was on the stack
* increase stack exactly by one, and leave the result of the expression on the stack