(SyntaxPattern (Unexpanded (\* 'c)) (Unexpanded (%\*% (NewName) 'c)))

Don’t expand when defining; don’t expand var c when substituting

(SyntaxPattern (%\*% 'n 'c) (Unexpanded (Rule 'n (/ (+ 'c 'n) ()))))

bla

(SyntaxPattern (tok 'c) (+ 'c WhiteSpace))

it shouldn’t matter what order expansion is done.

(Rule LanguageDeclaration (@$: "language" StringLiteral semicolon$))

(TermPattern [language : [string : 'v]] (Unexpanded [Import : 'v]))

Expand the @terms (expanding []) before defining,

Termpattern has two params, should both be expanded during expansion of termpattern. Some subtrees may be left unevaluated with “unexpanded”

Syntaxpattern has two params, should not be expanded during expansion of the Syntaxpattern term (at definition time)

When expanding, if a pattern match is found, the pattern variables should be bound to the unexpanded subtrees. There should be a way (specified with “unexpanded”) to enable (or prevent) expansion of the variable bindings

[SyntaxPattern [UnVar "pat"] [UnVar "repl"]] 🡪

[PatternPattern [Var "pat"] [Var "repl"]]

[SyntaxPattern [tok [Var c]] [+ [Var c] WhiteSpace]] 🡪

[PatternPattern [tok [Var c]] [+ [Var c] WhiteSpace]]

import java.util.\*;

public enum Operation {

plus("+") {

double eval(double x, double y) { return x + y; }

},

minus("-") {

double eval(double x, double y) { return x - y; }

},

times("\*") {

double eval(double x, double y) { return x \* y; }

},

divided\_by("/") {

double eval(double x, double y) { return x / y; }

};

private String opStr;

Operation(String opStr) {this.opStr = opStr;}

abstract double eval(double x, double y);

public static void main(String args[]) {

double x = Double.parseDouble(args[0]);

double y = Double.parseDouble(args[1]);

for (Operation op : Operation.values()) {

System.out.println(x + " " + op + " " + y + " = " + op.eval(x, y));

}

}

}

// Running this program produces the following output:

// java Operation 2.0 4.0

// 2.0 plus 4.0 = 6.0

// 2.0 minus 4.0 = -2.0

// 2.0 times 4.0 = 8.0

// 2.0 divided\_by 4.0 = 0.5

////////// the code this should translate into /////////////

// Typesafe enum with behaviors attached to constants

public abstract class Operation extends Enum<Operation> {

// Perform arithmetic operation represented by this constant

abstract double eval(double x, double y);

public static final Operation plus = new Operation("plus", 0, "+") {

double eval(double x, double y) { return x + y; }

};

public static final Operation minus = new Operation("minus", 1, "-") {

double eval(double x, double y) { return x - y; }

};

public static final Operation times = new Operation("times", 2, "\*") {

double eval(double x, double y) { return x \* y; }

};

public static final Operation divided\_by = new Operation("divided\_by", 3, "/") {

double eval(double x, double y) { return x / y; }

};

private static final Operation[] $VALUES = { plus, minus, times, divided\_by };

public static Operation[] values() {

return (Operation[]) $VALUES.clone();

}

public static Operation valueOf(String name) {

return (Operation) java.lang.Enum.valueOf(Operation.class, name);

}

private String opStr;

protected Operation(String name, int ordinal, String opStr) {

super(name, ordinal);

this.opStr = opStr;

}

}