Declarative Semantics Definition

Term Rewriting

Guido Wachsmuth

TU Delft
Delft University of Technology

Challenge the future

Course IN4303 2012/13
Compiler Construction

Assessment

last lecture

Assessment

last lecture

What are the plagues of non-declarative syntax definitions?

Assessment
last lecture

What are the plagues of non-declarative syntax definitions?

- grammar classes

Assessment

last lecture

What are the plagues of non-declarative syntax definitions?

- grammar classes
- disambiguation

Assessment

last lecture

What are the plagues of non-declarative syntax definitions?

- grammar classes
- disambiguation
- lexical syntax

Assessment

last lecture

What are the plagues of non-declarative syntax definitions?

- grammar classes
- disambiguation
- lexical syntax
- tree construction

Assessment
last lecture

What are the plagues of non-declarative syntax definitions?

- grammar classes
- disambiguation
- lexical syntax
- tree construction
- evolution

Assessment

last lecture

What are the plagues of non-declarative syntax definitions?

- grammar classes
- disambiguation
- lexical syntax
- tree construction
- evolution
- composition

Assessment

last lecture

What are the plagues of non-declarative syntax definitions?

- grammar classes
- disambiguation
- lexical syntax
- tree construction
- evolution
- composition
- restriction to parsers

Overview
today's lecture

manual implementation

- static analysis & error checking
- code generation
- semantic editor services

domain-specific language

- Stratego
- transform ASTs
- working on ATerms
- declarative style

I

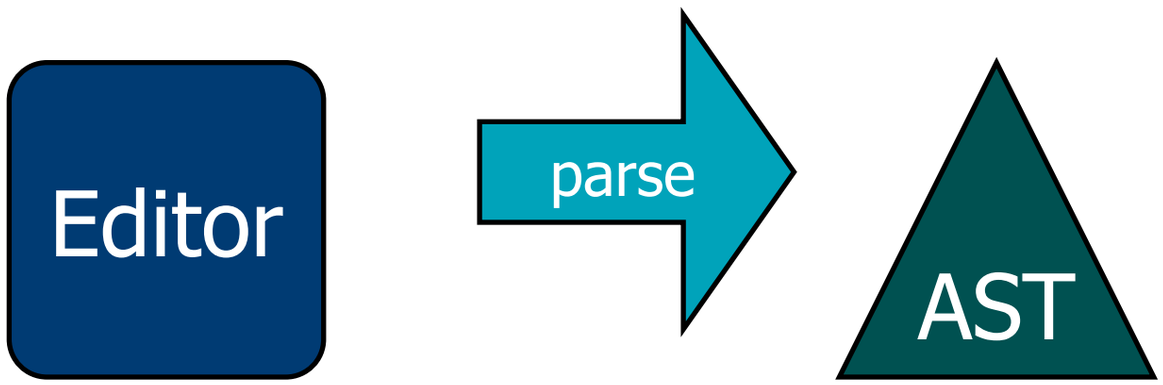
overview

Modern Compilers in IDEs

architecture

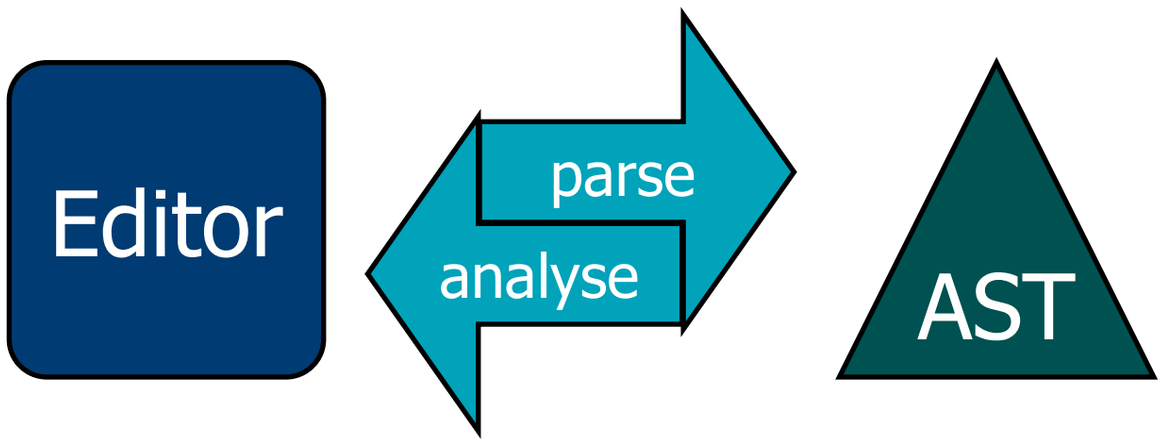
Modern Compilers in IDEs

architecture



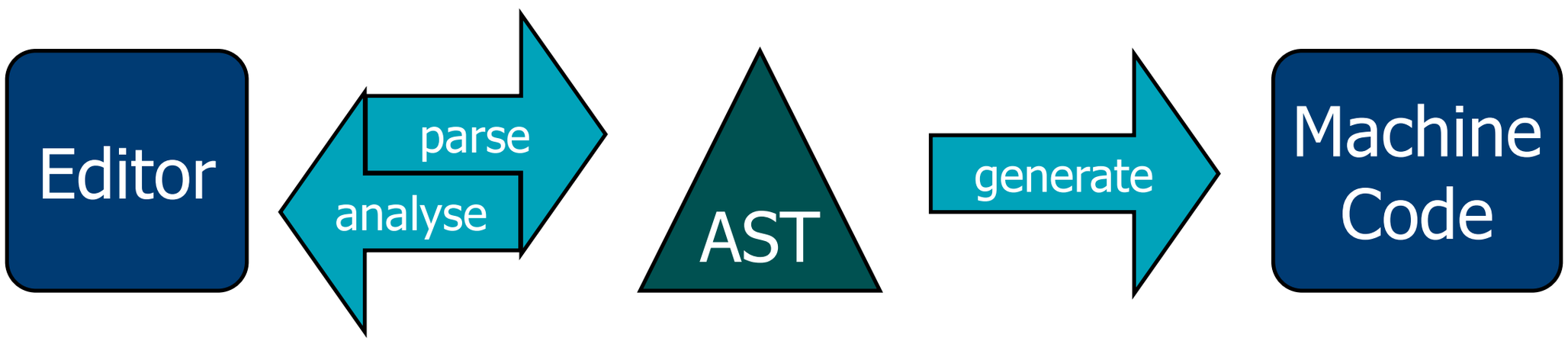
Modern Compilers in IDEs

architecture



Modern Compilers in IDEs

architecture



Recap: Compiler Construction

strategies

Recap: Compiler Construction
strategies

manual implementation

- general-purpose language, particular implementation techniques
- domain-specific language, domain = language processing

Recap: Compiler Construction
strategies

manual implementation

- general-purpose language, particular implementation techniques
- domain-specific language, domain = language processing

generation + generation

- input: language definition
- output: implementation in general-purpose language

Recap: Compiler Construction
strategies

manual implementation

- general-purpose language, particular implementation techniques
- domain-specific language, domain = language processing

generation + generation

- input: language definition
- output: implementation in general-purpose language

generation + interpretation

- virtual machine for language processing
- input: language definition
- output: “machine code”

Spoofax Language Workbench

generation + interpretation

Spoofax Language Workbench

generation + generation

Spoofax Language Workbench
manual implementation

Stratego

- declarative DSL
- domain: program transformation
- means to manipulate ASTs
- working on ATerms
- compiles to C or Java

Stratego in Spoofax

- static analysis & error checking
- code generation
- semantic editor services

Stratego

concepts

signatures

rewrite rules

- transform term to term
- left-hand side
  - pattern matching & variable binding
- right-hand side
  - pattern instantiation & variable substitution

rewrite strategies

- algorithm for applying rewrite rules

Stratego

example

```
module desugar

imports

  include/Tiger
  operators

strategies

  desugar-all = innermost(desugar)

rules

  desugar: IfThen(e1, e2) -> IfThenElse(e1, e2, NoVal())
```

II

signatures

Signatures

```
context-free syntax

   "-" Exp      -> Exp {cons("Uminus")}
   Exp "+"  Exp -> Exp {cons("Plus")}
   Exp "-"  Exp -> Exp {cons("Minus")}
   Exp "*"  Exp -> Exp {cons("Times")}
   Exp "/"  Exp -> Exp {cons("Divide")}

   Exp "="  Exp -> Exp {cons("Eq")}
   Exp "<>" Exp -> Exp {cons("Neq")}
   Exp ">"  Exp -> Exp {cons("Gt")}
   Exp "<"  Exp -> Exp {cons("Lt")}
   Exp ">=" Exp -> Exp {cons("Geq")}
   Exp "<=" Exp -> Exp {cons("Leq")}

   Exp "&"  Exp -> Exp {cons("And")}
   Exp "|"  Exp -> Exp {cons("Or")}
```

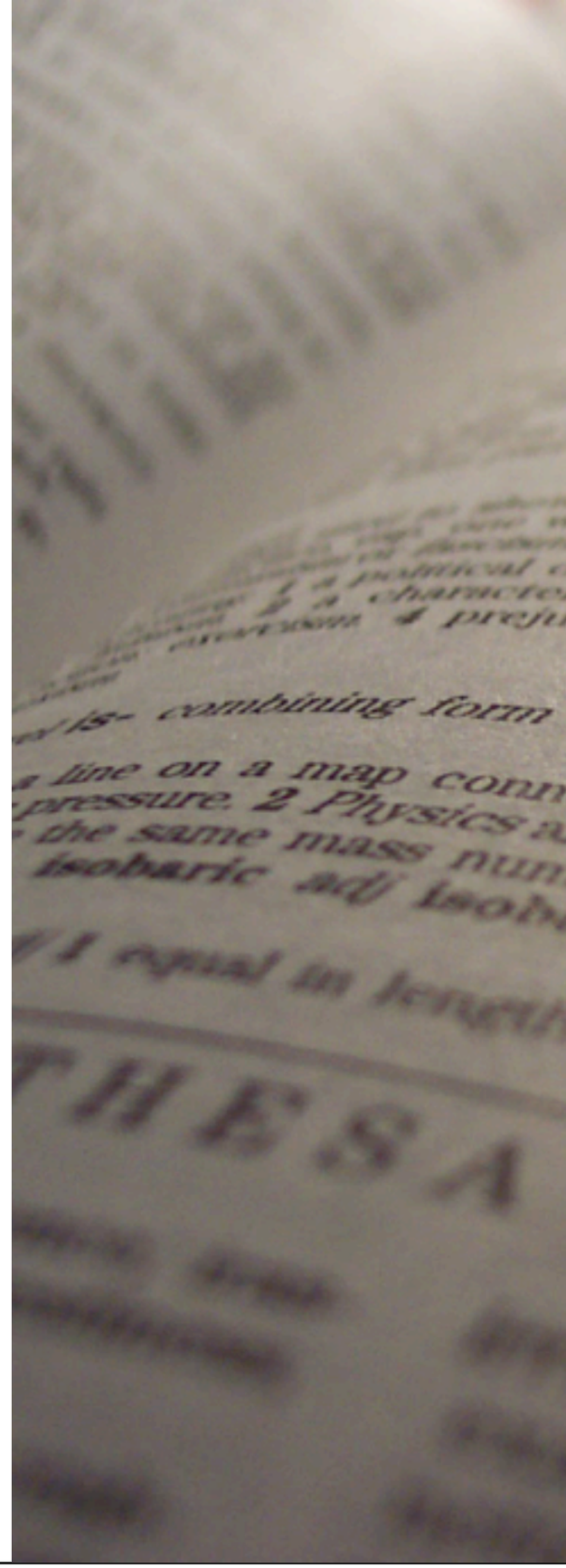
```
signature
   constructors
      Uminus : Exp       -> Exp
      Plus   : Exp * Exp -> Exp
      Minus  : Exp * Exp -> Exp
      Times  : Exp * Exp -> Exp
      Divide : Exp * Exp -> Exp

      Eq     : Exp * Exp -> Exp
      Neq    : Exp * Exp -> Exp
      Gt     : Exp * Exp -> Exp
      Lt     : Exp * Exp -> Exp
      Geq    : Exp * Exp -> Exp
      Leq    : Exp * Exp -> Exp

      And    : Exp * Exp -> Exp
      Or     : Exp * Exp -> Exp
```

ATerms in Stratego

signature

```
signature
   constructors
      Constructor : Cons                -> Term
      Application : Cons * List(Term)   -> Term
      List        : List(Term)          -> Term
      Tuple       : List(Term)          -> Term
```

```
signature
   constructors
      Nil  :                      -> List(a)
      Cons : a * List(a)          -> List(a)
      Conc : List(a) * List(a)    -> List(a)
```



III

rewrite rules

Desugaring

```
if x then
   printint(x)
```

```
if x then
   printint(x)
else
   ()
```

Desugaring

```
if x then
   printint(x)
```

```
if x then
   printint(x)
else
   ()
```

```
IfThen(
  Var("x")
, Call(
    "printint"
  , [Var("x")]
  )
)
```

```
IfThenElse(
  Var("x")
, Call(
    "printint"
  , [Var("x")]
  )
, NoVal()
)
```

Desugaring

```
if x then
   printint(x)
```

```
if x then
   printint(x)
else
   ()
```

```
IfThen(
  Var("x")
, Call(
    "printint"
  , [Var("x")]
  )
)
```

```
IfThenElse(
  Var("x")
, Call(
    "printint"
  , [Var("x")]
  )
, NoVal()
)
```

```
desugar: IfThen(e1, e2) -> IfThenElse(e1, e2, NoVal())
```

Desugaring

```
if x then
   printint(x)
```

```
if x then
   printint(x)
else
   ()
```

```
IfThen(
  Var("x")
, Call(
    "printint"
  , [Var("x")]
  )
)
```

pattern matching

```
IfThenElse(
  Var("x")
, Call(
    "printint"
  , [Var("x")]
  )
, NoVal()
)
```

```
desugar: IfThen(e1, e2) -> IfThenElse(e1, e2, NoVal())
```

Desugaring

```
if x then
   printint(x)
```

```
if x then
   printint(x)
else
   ()
```

```
IfThen(
   Var("x")
 , Call(
     "printint"
   , [Var("x")]
   )
)
```

e1

e2

pattern matching

```
IfThenElse(
   Var("x")
 , Call(
     "printint"
   , [Var("x")]
   )
 , NoVal()
)
```

```
desugar: IfThen(e1, e2) -> IfThenElse(e1, e2, NoVal())
```

Desugaring

```
if x then
   printint(x)
```

```
if x then
   printint(x)
else
   ()
```

```
IfThen(
   Var("x")
,  Call(
      "printint"
   ,  [Var("x")]
   )
)
```

e1

e2

pattern matching

```
IfThenElse(
   Var("x")
,  Call(
      "printint"
   ,  [Var("x")]
   )
,  NoVal()
)
```

pattern instantiation

```
desugar: IfThen(e1, e2) -> IfThenElse(e1, e2, NoVal())
```

Desugaring

```
if x then
   printint(x)
```

```
if x then
   printint(x)
else
   ()
```

```
IfThen(
   Var("x")
 , Call(
     "printint"
   , [Var("x")]
   )
)
```

pattern matching

```
IfThenElse(
   Var("x")
 , Call(
     "printint"
   , [Var("x")]
   )
 , NoVal()
)
```

pattern instantiation

e1

e2

```
desugar: IfThen(e1, e2) -> IfThenElse(e1, e2, NoVal())
```

More Desugaring

```
desugar: Uminus(e)      -> Bop(MINUS(), Int("0"), e)


desugar: Plus(e1, e2)   -> Bop(PLUS(), e1, e2)
desugar: Minus(e1, e2)  -> Bop(MINUS(), e1, e2)
desugar: Times(e1, e2)  -> Bop(MUL(), e1, e2)
desugar: Divide(e1, e2) -> Bop(DIV(), e1, e2)


desugar: Eq(e1, e2)  -> Rop(EQ(), e1, e2)
desugar: Neq(e1, e2) -> Rop(NE(), e1, e2)
desugar: Leq(e1, e2) -> Rop(LE(), e1, e2)
desugar: Lt(e1, e2)  -> Rop(LT(), e1, e2)
desugar: Geq(e1, e2) -> Rop(LE(), e2, e1)
desugar: Gt(e1, e2)  -> Rop(LT(), e2, e1)


desugar: And(e1, e2) -> IfThenElse(e1, e2, Int("0"))
desugar: Or(e1, e2)  -> IfThenElse(e1, Int("1"), e2)
```

```
signature constructors
  PLUS:   BinOp
  MINUS:  BinOp
  MUL:    BinOp
  DIV:    BinOp

  EQ:  RelOp
  NE:  RelOp
  LE:  RelOp
  LT:  RelOp

  Bop: BinOp * Expr * Expr -> Expr
  Rop: RelOp * Expr * Expr -> Expr
```

Constant Folding

```
x := 21 + 21 + x
```

```
x := 42 + x
```

Constant Folding

```
x := 21 + 21 + x
```

```
x := 42 + x
```

```
Assign(
  Var("x")
, Plus(
    Plus(
      Int("21")
    , Int("21")
    )
  , Var("x")
  )
)
```

```
Assign(
  Var("x")
, Plus(
    Int("42")
  , Var("x")
  )
)
```

Constant Folding

```
x := 21 + 21 + x
```

```
x := 42 + x
```

```
Assign(
  Var("x")
, Plus(
    Plus(
      Int("21")
    , Int("21")
    )
  , Var("x")
  )
)
```

```
Assign(
  Var("x")
, Plus(
    Int("42")
  , Var("x")
  )
)
```

```
eval: Bop(PLUS(), Int(i1), Int(i2)) -> Int(i3)
        where <addS> (i1, i2) => i3
```

More Constant Folding

```
eval: Bop(PLUS(), Int(i1), Int(i2))  -> Int(<addS> (i1, i2))

eval: Bop(MINUS(), Int(i1), Int(i2)) -> Int(<subtS> (i1, i2))

eval: Bop(MUL(), Int(i1), Int(i2))   -> Int(<mulS> (i1, i2))

eval: Bop(DIV(), Int(i1), Int(i2))   -> Int(<divS> (i1, i2))

eval: Rop(EQ(), Int(i), Int(i))   -> Int("1")
eval: Rop(EQ(), Int(i1), Int(i2)) -> Int("0") where not( <eq> (i1, i2) )

eval: Rop(NE(), Int(i), Int(i))   -> Int("0")
eval: Rop(NE(), Int(i1), Int(i2)) -> Int("1") where not( <eq> (i1, i2) )

eval: Rop(LT(), Int(i1), Int(i2)) -> Int("1") where <ltS> (i1, i2)
eval: Rop(LT(), Int(i1), Int(i2)) -> Int("0") where not( <ltS> (i1, i2) )

eval: Rop(LE(), Int(i1), Int(i2)) -> Int("1") where <leqS> (i1, i2)
eval: Rop(LE(), Int(i1), Int(i2)) -> Int("0") where not( <leqS> (i1, i2))
```

Rewrite Rules

parameters

```
f(x,y|a,b): lhs -> rhs
```

- strategy or rule parameters `x,y`
- term parameters `a,b`
  - no matching

```
f(|a,b): lhs -> rhs
```

- optional strategy parameters

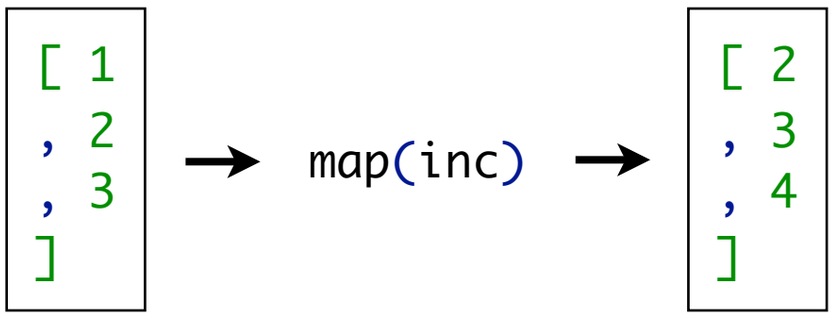
```
f(x,y): lhs -> rhs
```

- optional term parameters

```
f: lhs -> rhs
```

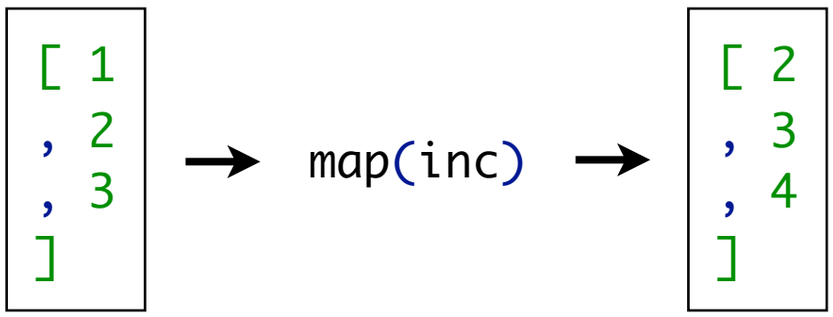
Rules with Parameters

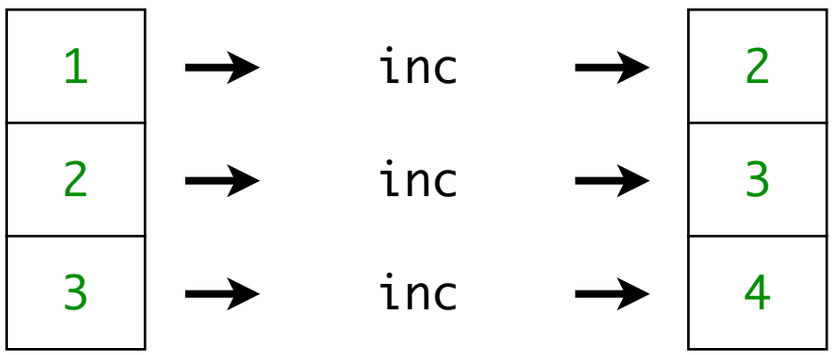
example: map



Rules with Parameters

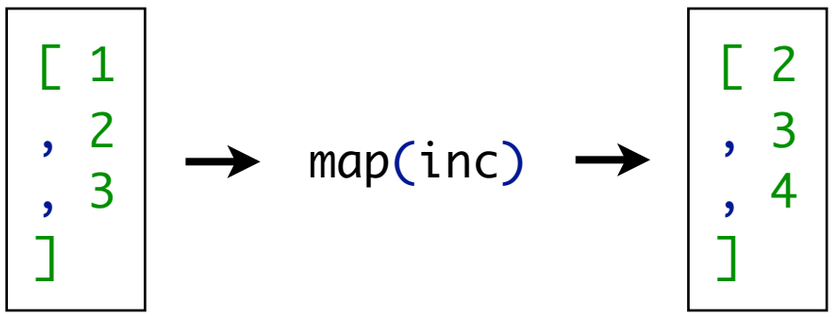
example: map





Rules with Parameters

example: map

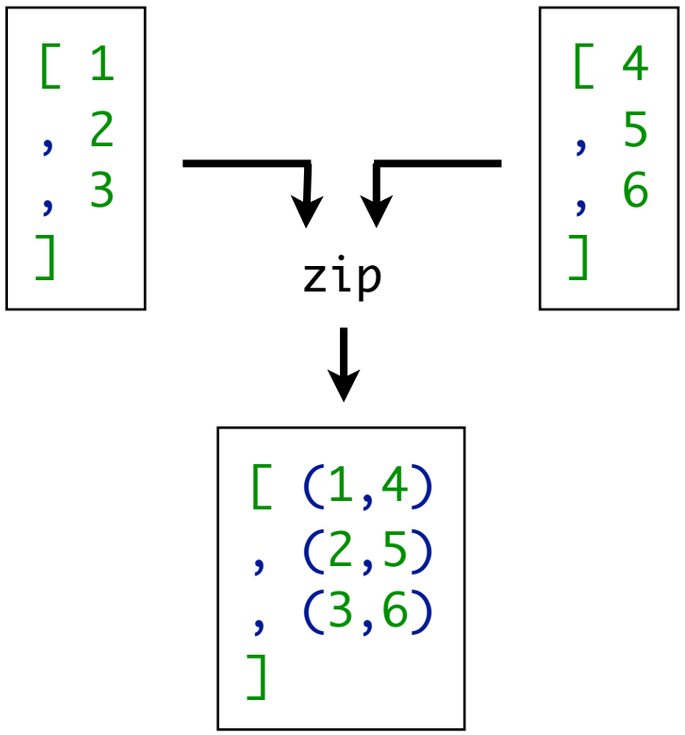


1	→	inc	→	2
2	→	inc	→	3
3	→	inc	→	4

```
map(s): []     -> []
map(s): [x|xs] -> [<s> x | <map(s)> xs]
```

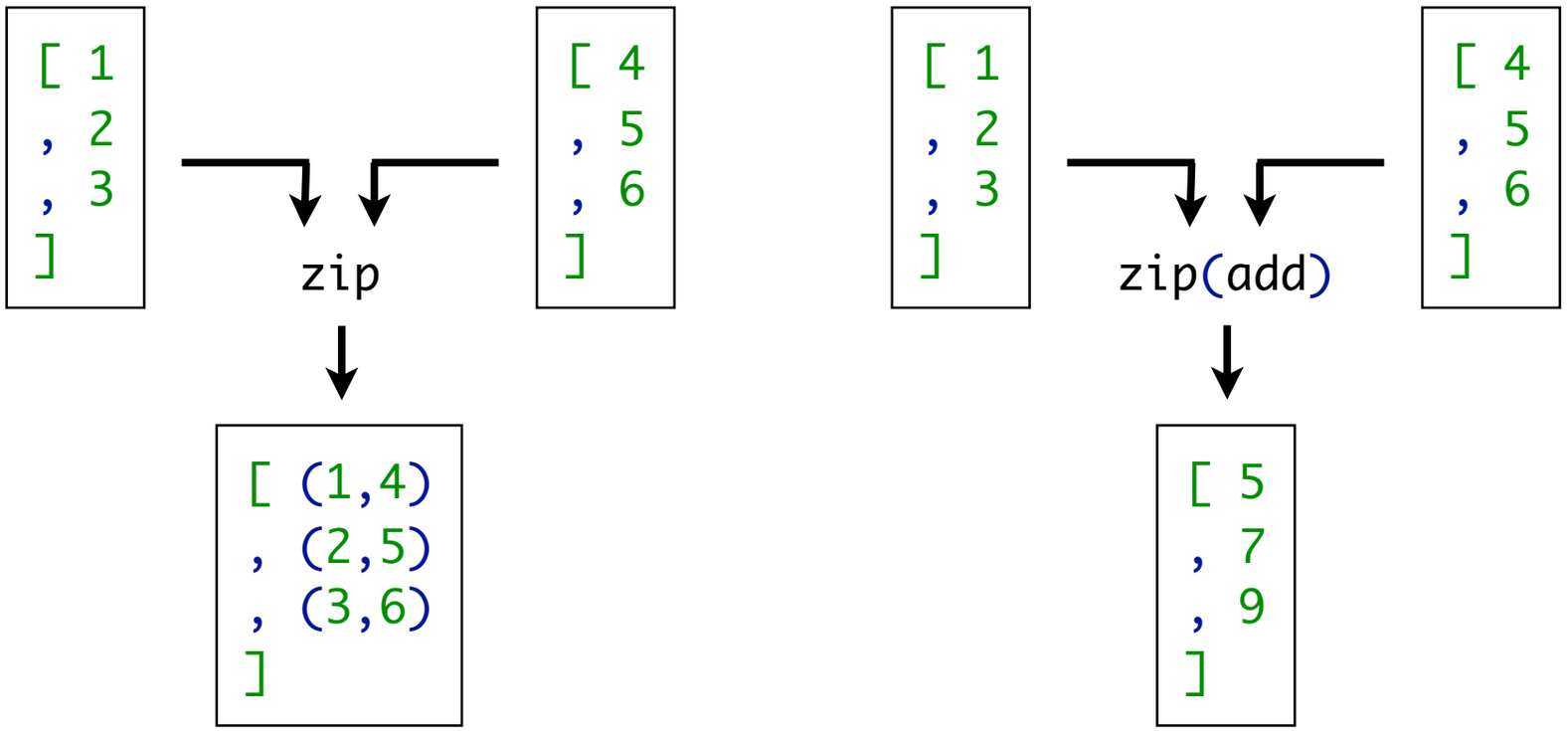
Rules with Parameters

example: zip



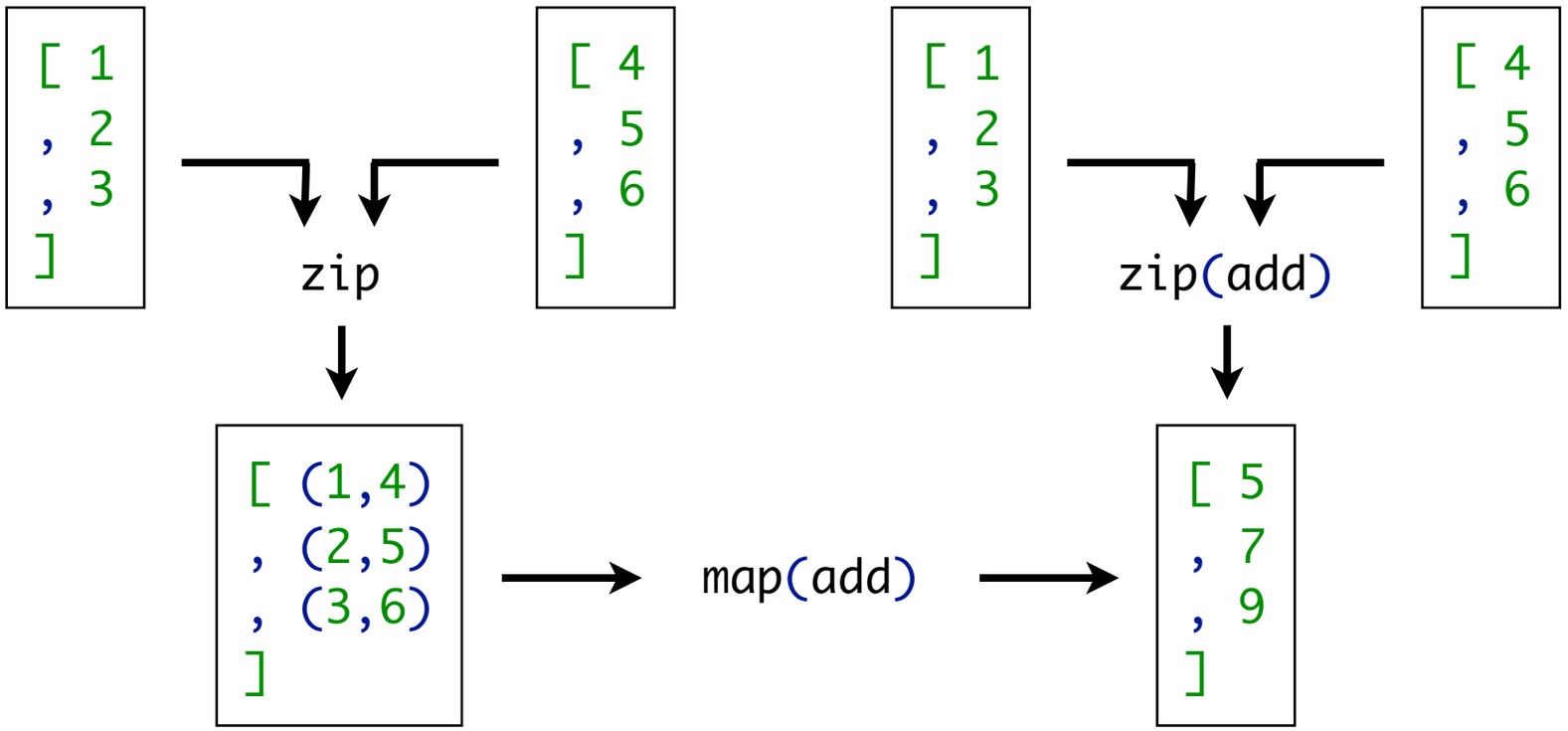
Rules with Parameters

example: zip



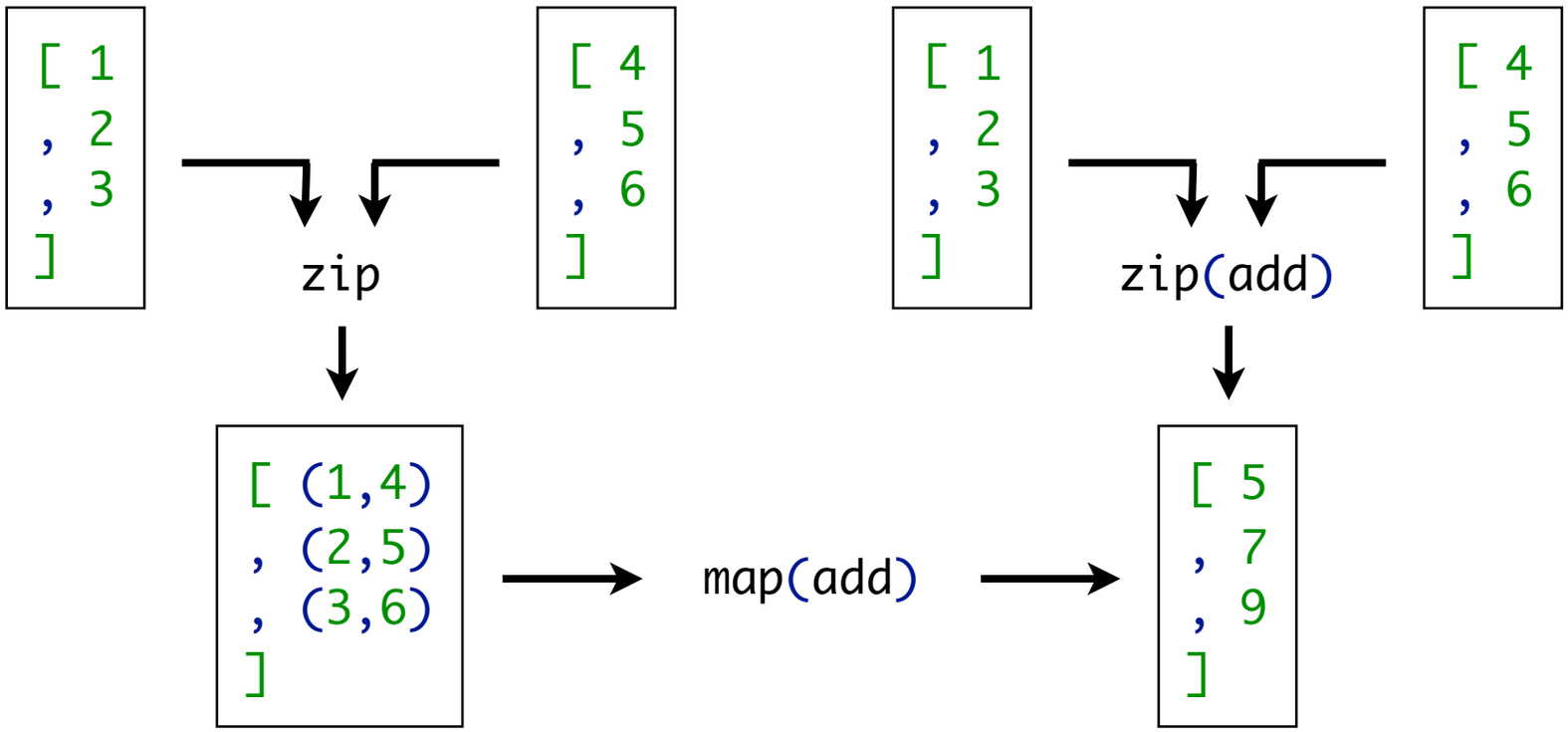
Rules with Parameters

example: zip



Rules with Parameters

example: zip

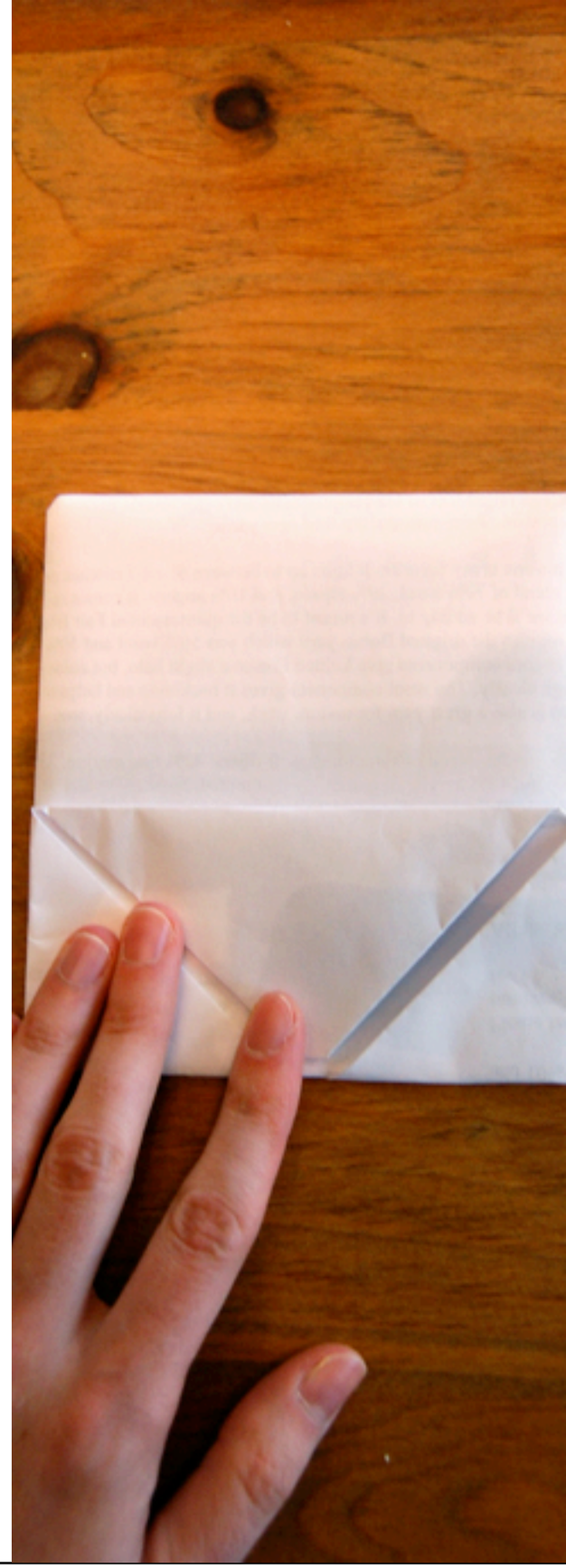



```
zip(s): ([],[])          -> []
zip(s): ([x|xs],[y|ys]) -> [<s> (x,y) | <zip(s)> (xs,ys)]

zip = zip(id)
```

Rules with Parameters

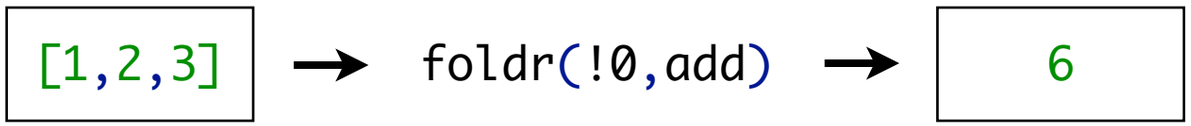
example: fold

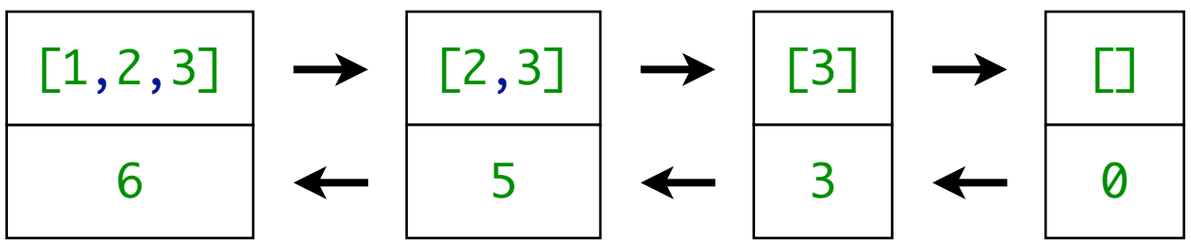
```
[1,2,3]  →  foldr(!0,add)  →  6
```

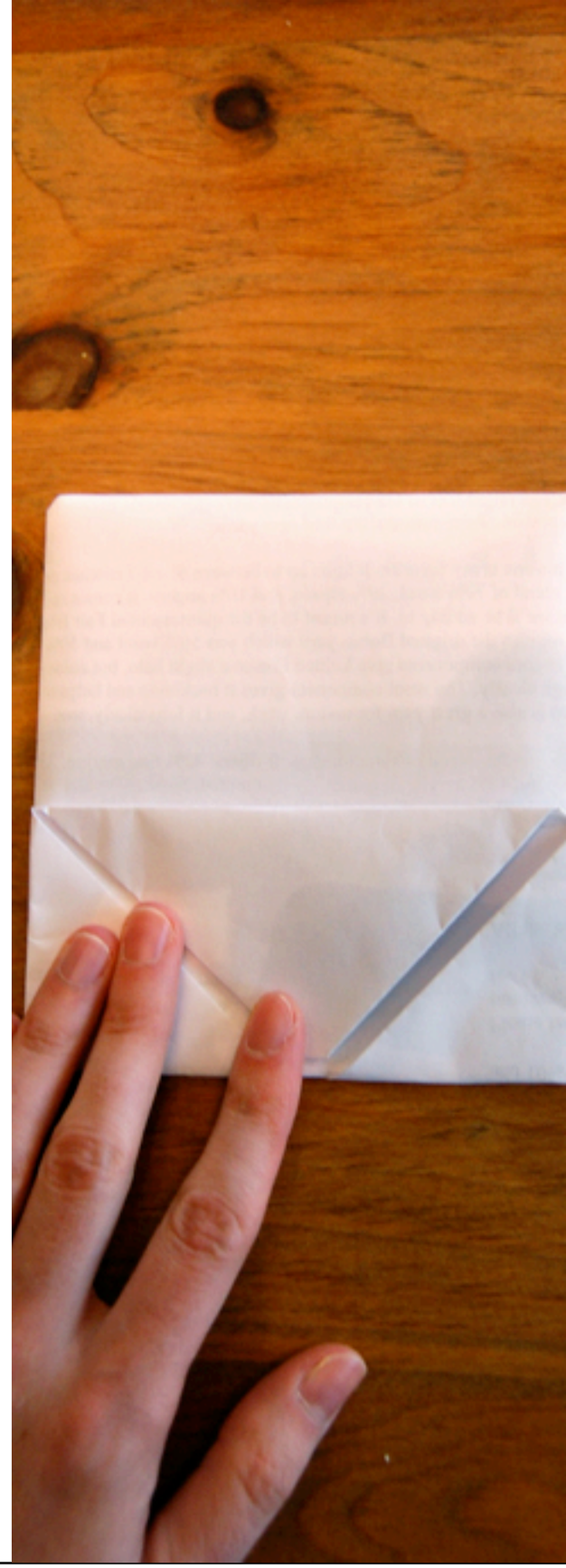


Rules with Parameters

example: fold

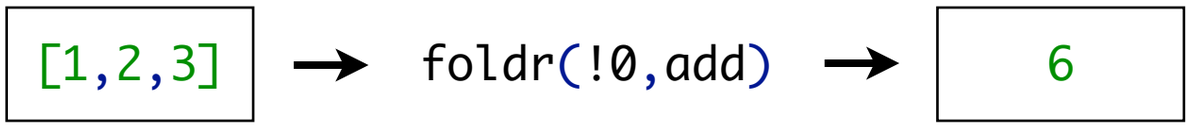


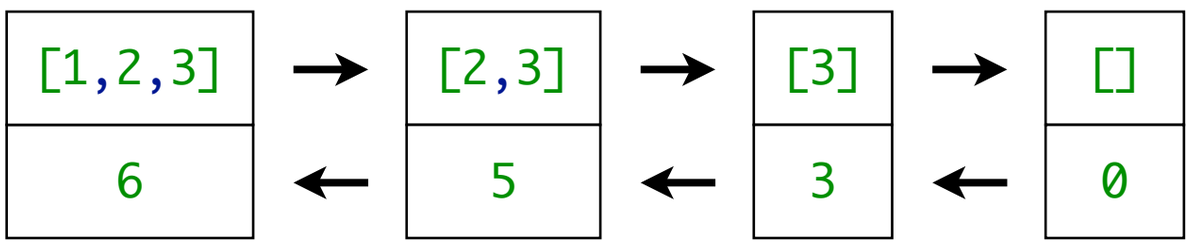




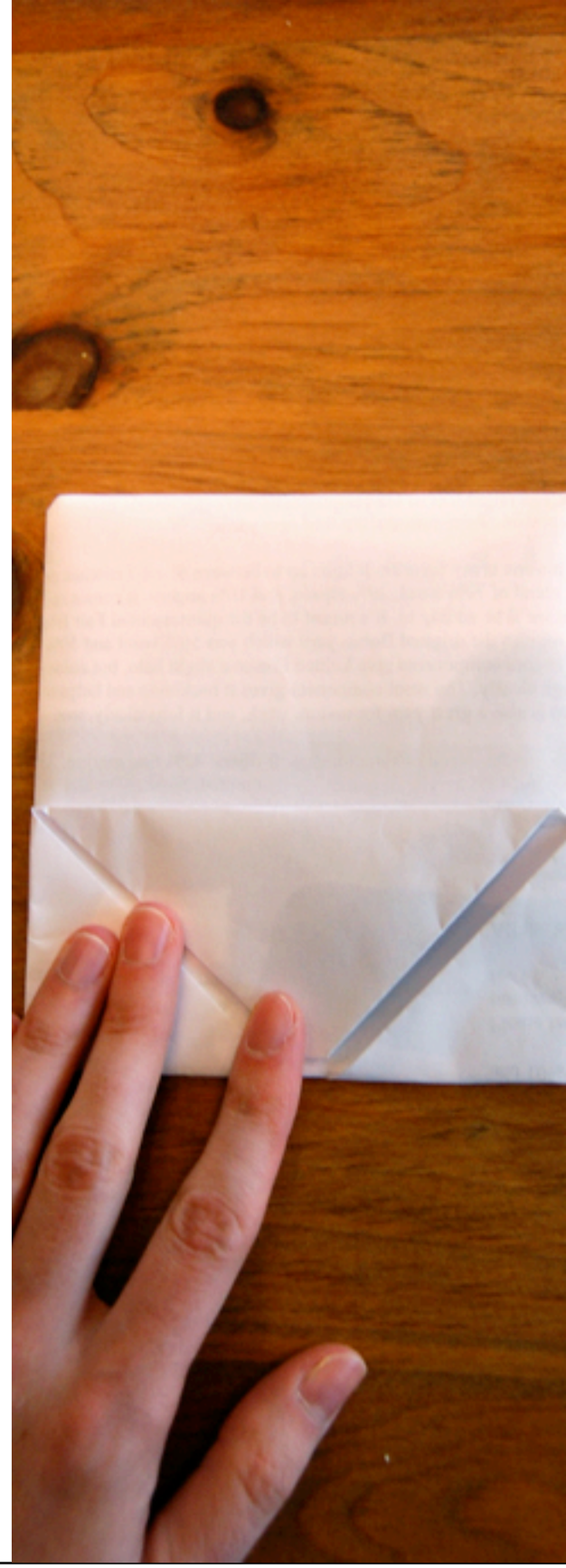
Rules with Parameters

example: fold






```
foldr(s1,s2): []     -> <s1>
foldr(s1,s2): [x|xs] -> <s2> (x,<foldr(s1,s2)> xs)
```



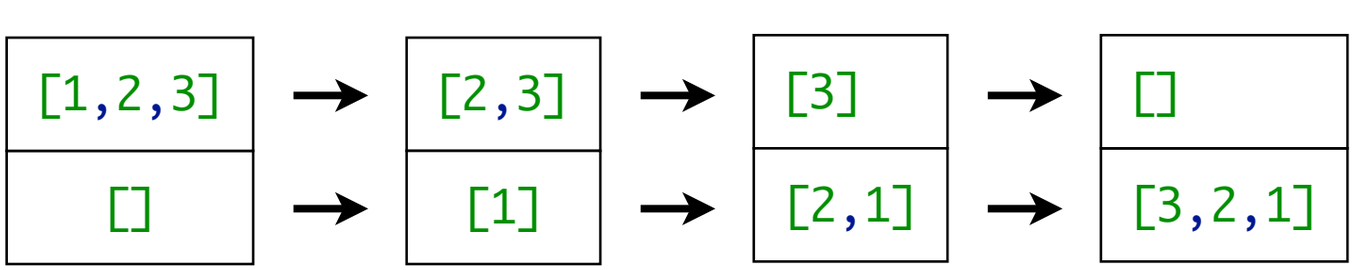
Rules with Parameters

example: inverse

```
[1,2,3]  →  inverse(|[])  →  [3,2,1]
```

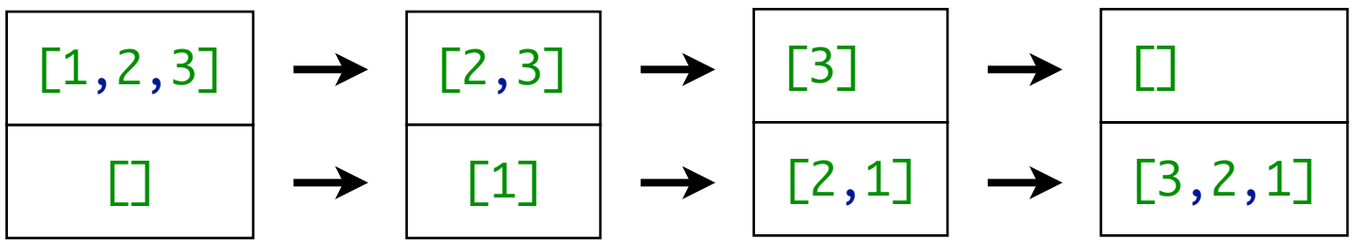
Rules with Parameters

example: inverse



Rules with Parameters

example: inverse




```
inverse(|is): []     -> is
inverse(|is): [x|xs] -> <inverse(|[x|is])> xs
```

coffee break

IV

rewrite strategies

Transformation Definitions

rules and strategies

rewrite rules

- term to term
- left-hand side matching
- right-hand side instantiation
- conditional
- partial transformation

rewrite strategies

- rule selection
- algorithm
- composition of transformations

Stratego

example

```
module desugar

imports

  include/Tiger
  operators

strategies

  desugar-all = innermost(desugar)

rules

  desugar: IfThen(e1, e2) -> IfThenElse(e1, e2, NoVal())
```

Stratego

example

```
module eval

imports
  include/Tiger
  operators
  desugar

strategies

  eval-all = innermost(desugar + eval)

rules
  eval: Bop(PLUS(),Int(i1),Int(i2)) -> Int(i3)
          where <addS> (i1, i2) => i3
```

Strategy Combinators

identity

```
id
```

Strategy Combinators

identity

```
id
```

failure

```
fail
```

Strategy Combinators

identity

```
id
```

failure

```
fail
```

sequential composition

```
s1 ; s2
```

Strategy Combinators

identity

```
id
```

failure

```
fail
```

sequential composition

```
s1 ; s2
```

deterministic choice

```
s1 <+ s2
```

Strategy Combinators

identity

```
id
```

failure

```
fail
```

sequential composition

```
s1 ; s2
```

deterministic choice

```
s1 <+ s2
```

non-deterministic choice

```
s1 + s2
```

Strategy Combinators

identity

```
id
```

failure

```
fail
```

sequential composition

```
s1 ; s2
```

deterministic choice

```
s1 <+ s2
```

non-deterministic choice

```
s1 + s2
```

guarded choice

```
s1 < s2 + s3
```

Strategy Combinators

variables

pattern matching

```
?t
```

Strategy Combinators

variables

pattern matching

```
?t
```

pattern instantiation

```
!t
```

Strategy Combinators

variables

pattern matching

```
?t
```

pattern instantiation

```
!t
```

strategy application

```
<s> t ≡ !t ; s
```

Strategy Combinators

variables

pattern matching

```
?t
```

pattern instantiation

```
!t
```

strategy application

```
<s> t ≡ !t ; s
```

result matching

```
s => t ≡ s ; ?t
```

```
<s> t1 => t2
t2 := <s> t1
```

Strategy Combinators

variables

pattern matching

```
?t
```

pattern instantiation

```
!t
```

strategy application

```
<s> t ≡ !t ; s
```

result matching

```
s => t ≡ s ; ?t
```

```
<s> t1 => t2
```

```
t2 := <s> t1
```

variable scope

```
{x,y: s}
```

Strategy Combinators

rules and strategies

named rewrite rule

```
l: t1 -> t2 where s ≡  l = ?t1 ; s ; !t2
```

Strategy Combinators

rules and strategies

named rewrite rule

```
l: t1 -> t2 where s ≡  l = ?t1 ; s ; !t2
```

unscoped rewrite rule

```
(t1 -> t2 where s) ≡ ?t1 ; s ; !t2
```

Strategy Combinators

rules and strategies

named rewrite rule

```
l: t1 -> t2 where s ≡  l = ?t1 ; s ; !t2
```

unscoped rewrite rule

```
(t1 -> t2 where s) ≡ ?t1 ; s ; !t2
```

strategy definition

```
f(x,y|a,b) = s
```

Strategy Combinators

examples

```
try(s) = s <+ id

repeat(s) = try(s ; repeat(s))
```

Strategy Combinators

examples

```
try(s) = s <+ id

repeat(s) = try(s ; repeat(s))

topdown(s) = s ; all(topdown(s))
```

Strategy Combinators

examples

```
try(s) = s <+ id

repeat(s) = try(s ; repeat(s))

topdown(s) = s ; all(topdown(s))

alltd(s) = s <+ all(alltd(s))
```

Strategy Combinators

examples

```
try(s) = s <+ id

repeat(s) = try(s ; repeat(s))

topdown(s) = s ; all(topdown(s))

alltd(s) = s <+ all(alltd(s))

bottomup(s) = all(bottomup(s)) ; s

innermost(s) = bottomup(try(s ; innermost(s)))

oncetd(s) = s <+ one(oncetd(s))

contains(|t) = oncetd(?t)
```

Strategy Combinators

examples

```
try(s) = s <+ id

repeat(s) = try(s ; repeat(s))

topdown(s) = s ; all(topdown(s))

alltd(s) = s <+ all(alltd(s))

bottomup(s) = all(bottomup(s)) ; s

innermost(s) = bottomup(try(s ; innermost(s)))

oncetd(s) = s <+ one(oncetd(s))

contains(|t) = oncetd(?t)
```

Strategy Combinators

examples

```
try(s) = s <+ id

repeat(s) = try(s ; repeat(s))

topdown(s) = s ; all(topdown(s))

alltd(s) = s <+ all(alltd(s))

bottomup(s) = all(bottomup(s)) ; s

innermost(s) = bottomup(try(s ; innermost(s)))

oncetd(s) = s <+ one(oncetd(s))

contains(|t) = oncetd(?t)
```

Strategy Combinators

examples

```
try(s) = s <+ id

repeat(s) = try(s ; repeat(s))

topdown(s) = s ; all(topdown(s))

alltd(s) = s <+ all(alltd(s))

bottomup(s) = all(bottomup(s)) ; s

innermost(s) = bottomup(try(s ; innermost(s)))

oncetd(s) = s <+ one(oncetd(s))

contains(|t) = oncetd(?t)
```

Strategy Combinators

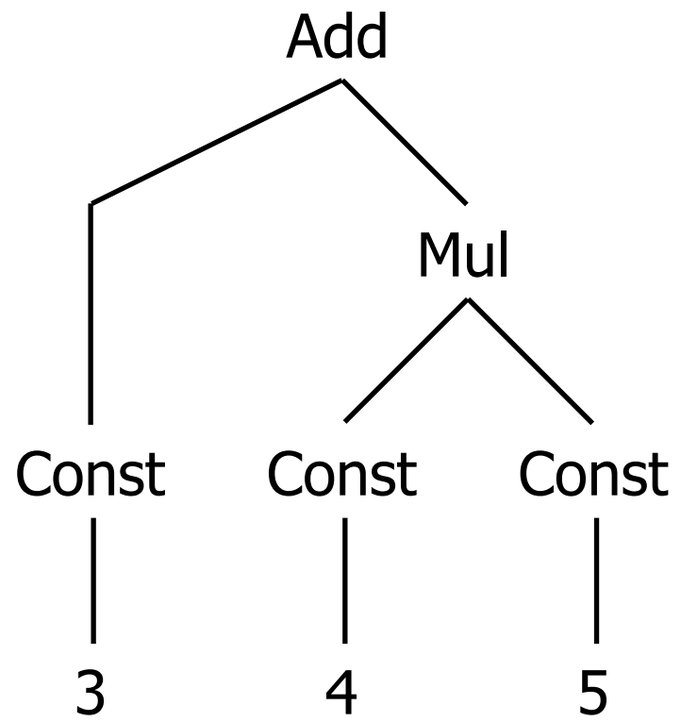
examples

```
try(s) = s <+ id

repeat(s) = try(s ; repeat(s))

topdown(s) = s ; all(topdown(s))

alltd(s) = s <+ all(alltd(s))

bottomup(s) = all(bottomup(s)) ; s

innermost(s) = bottomup(try(s ; innermost(s)))

oncetd(s) = s <+ one(oncetd(s))

contains(|t) = oncetd(?t)
```

Stratego

example

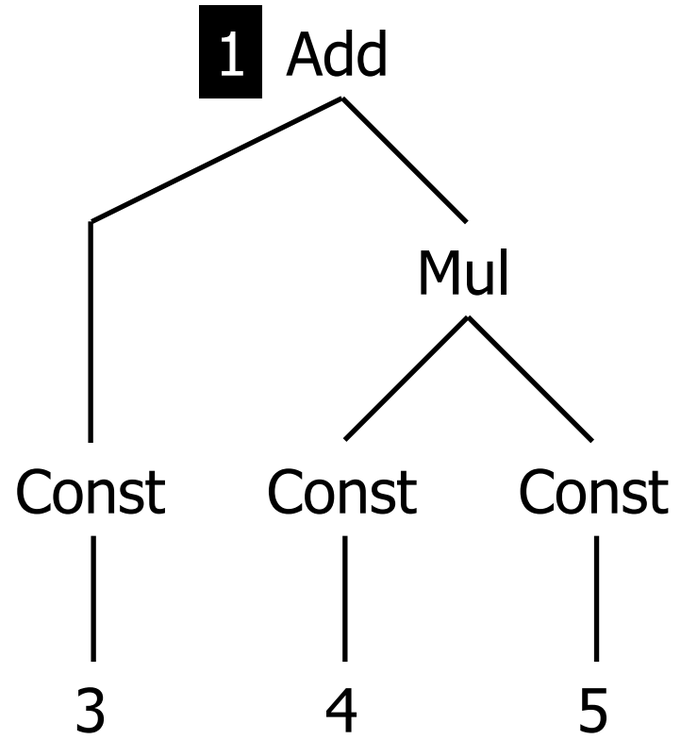
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(switch)
```



Stratego

example

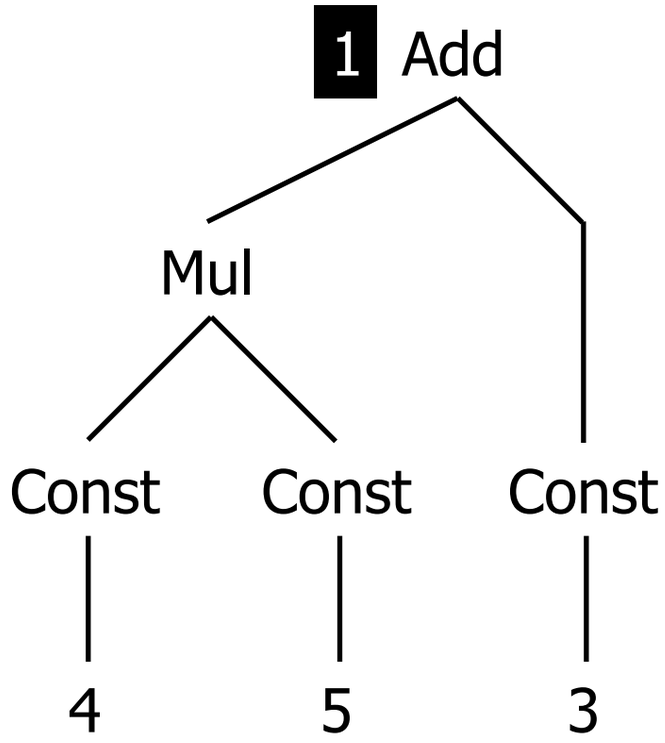
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))

topdown(switch)
```



Stratego

example

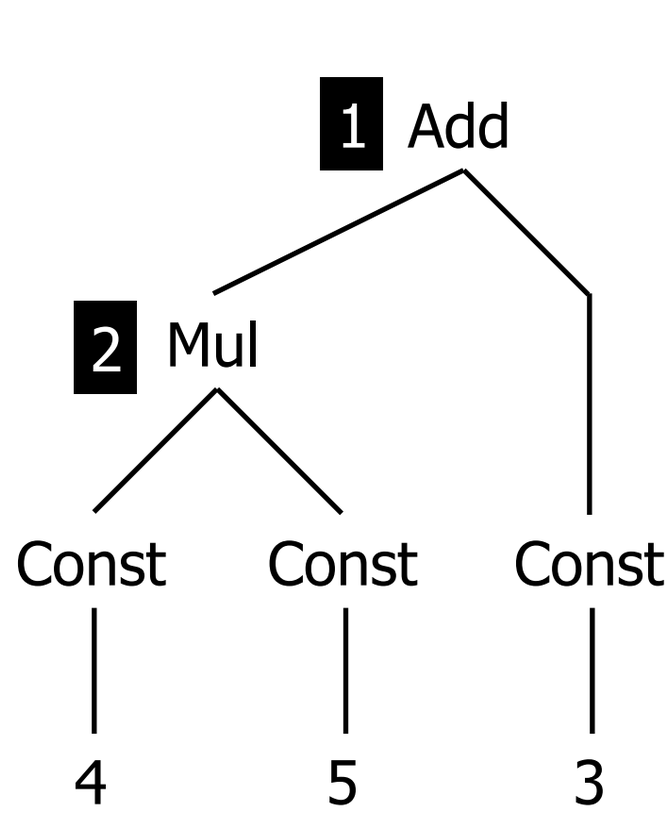
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))

topdown(switch)
```



Stratego

example

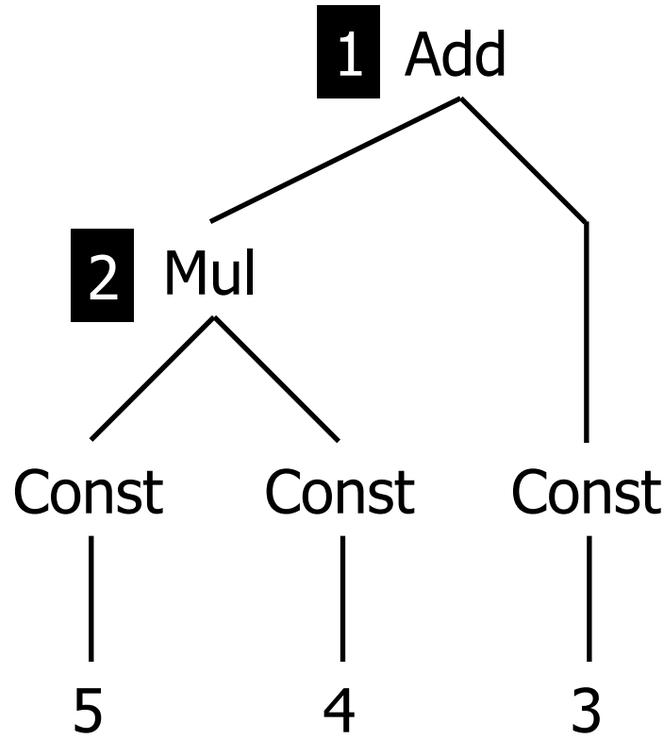
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))

topdown(switch)
```



Stratego

example

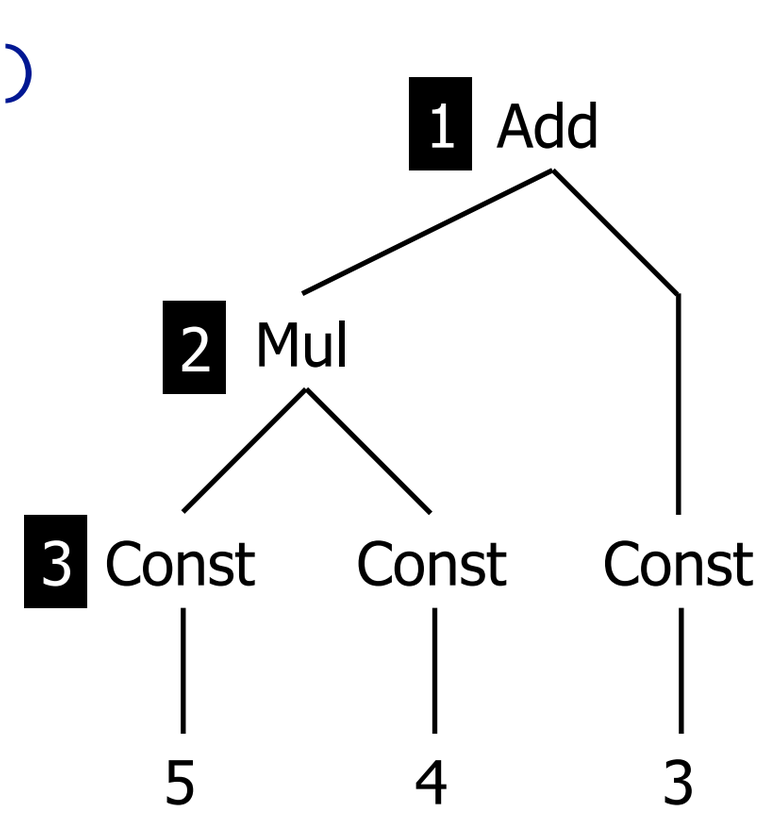
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))

topdown(switch)
```



Stratego

example

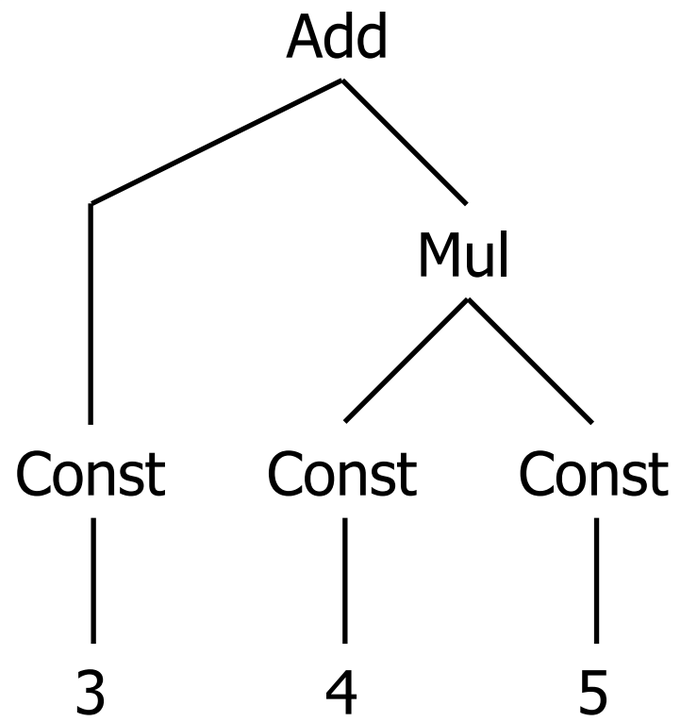
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))

topdown(switch)
```



Stratego

example

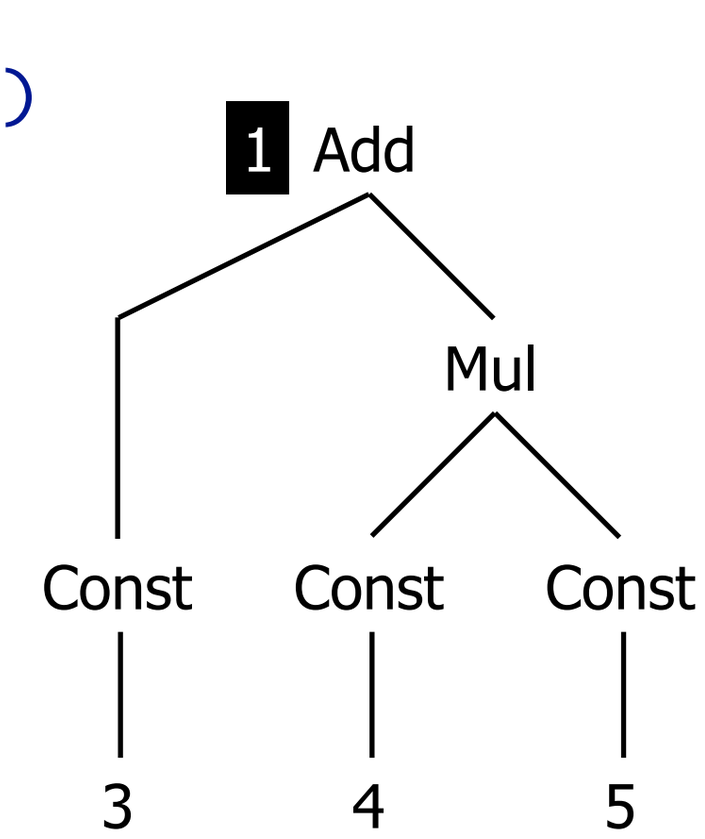
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

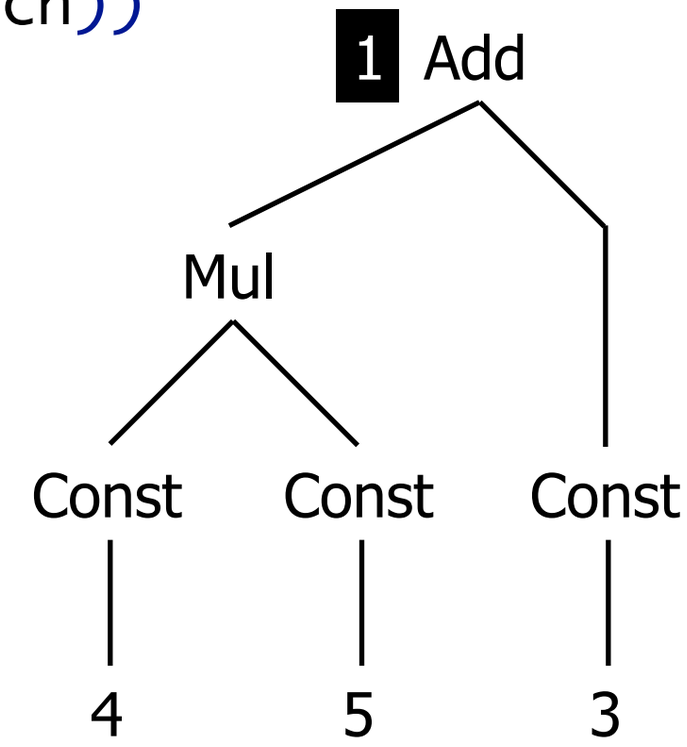
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

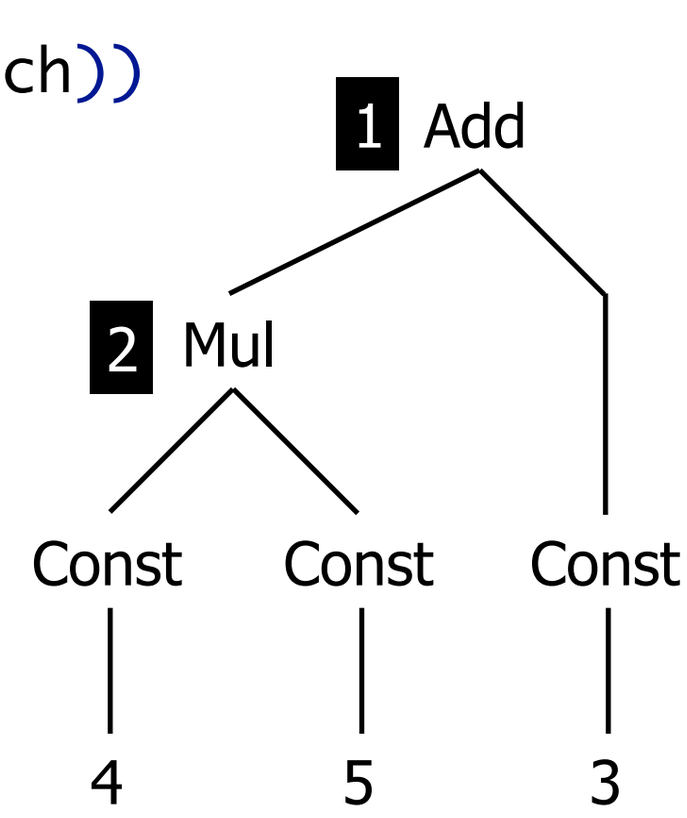
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

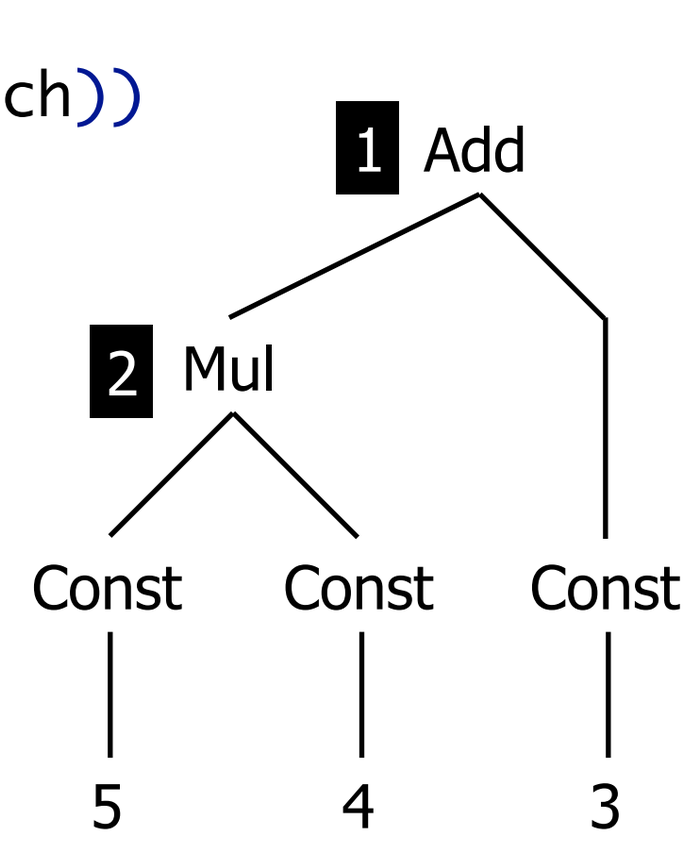
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

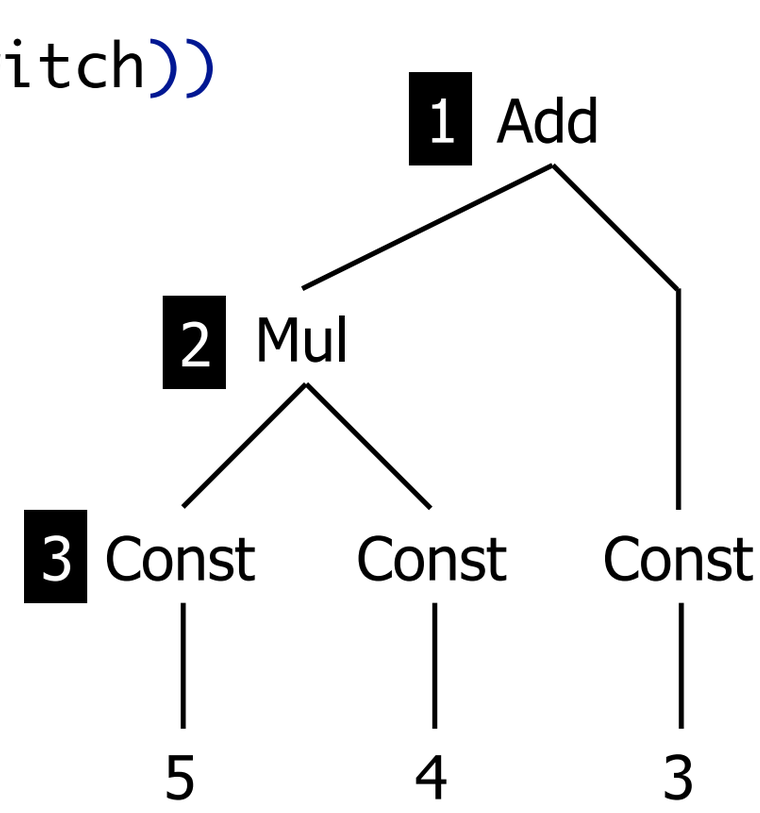
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

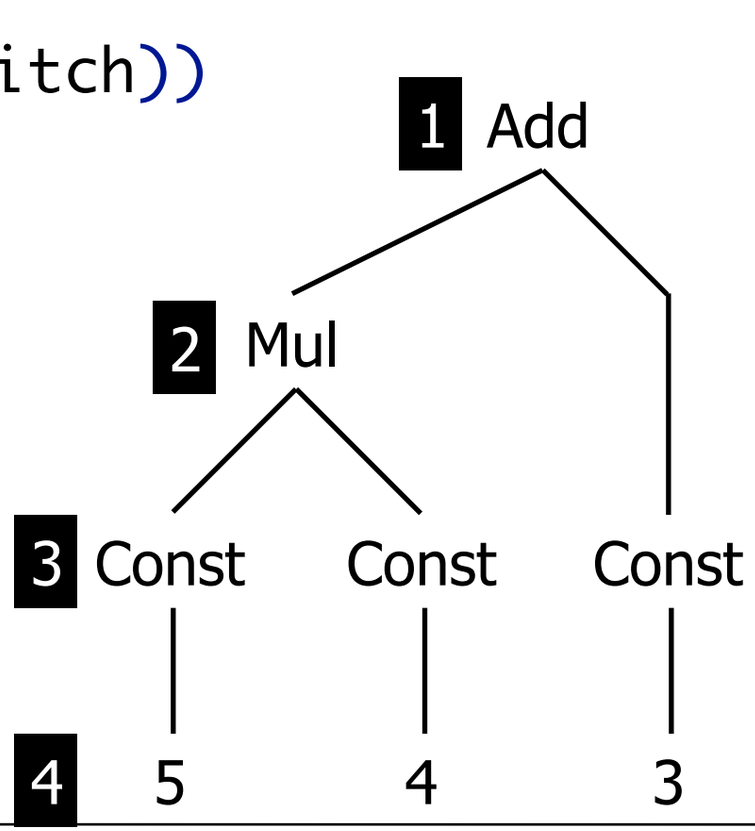
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

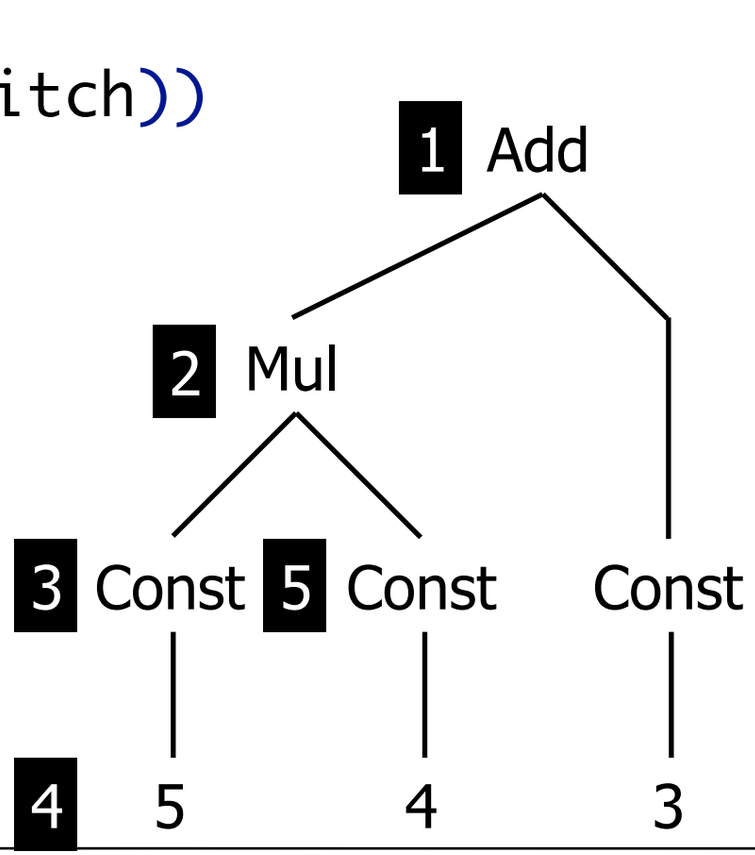
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

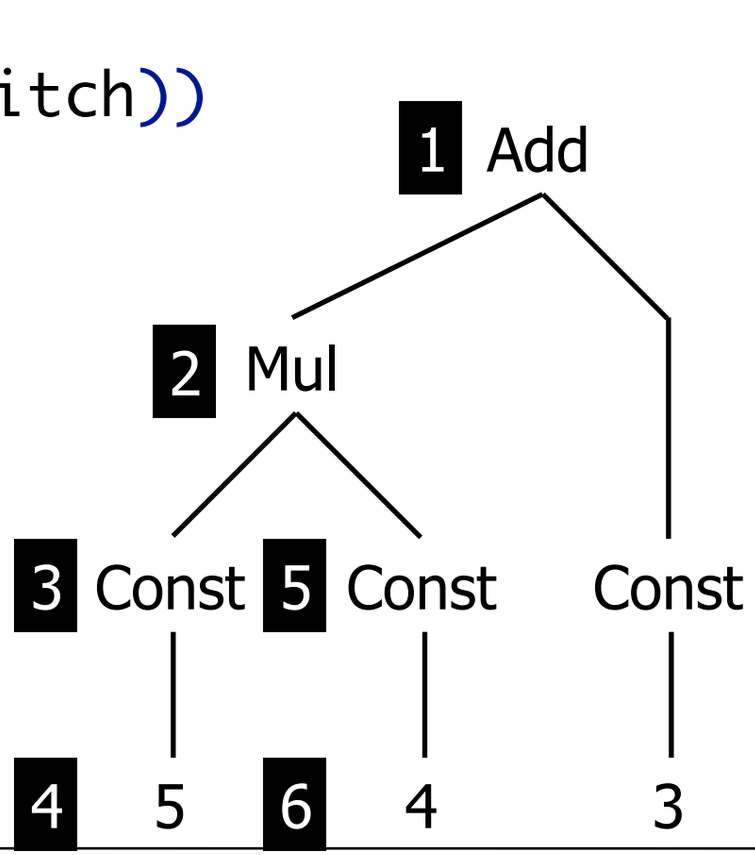
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

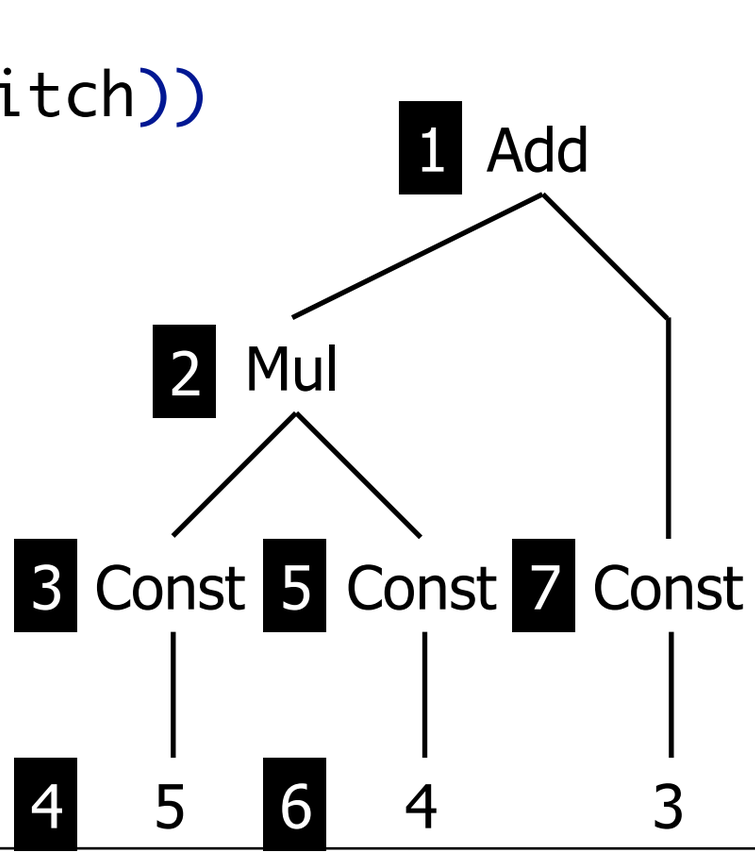
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

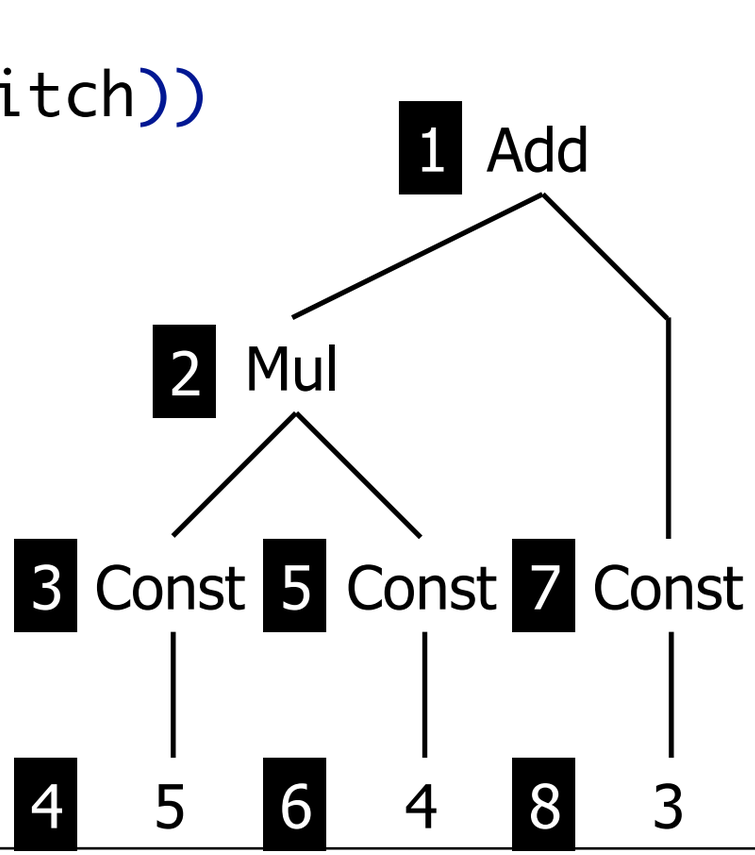
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

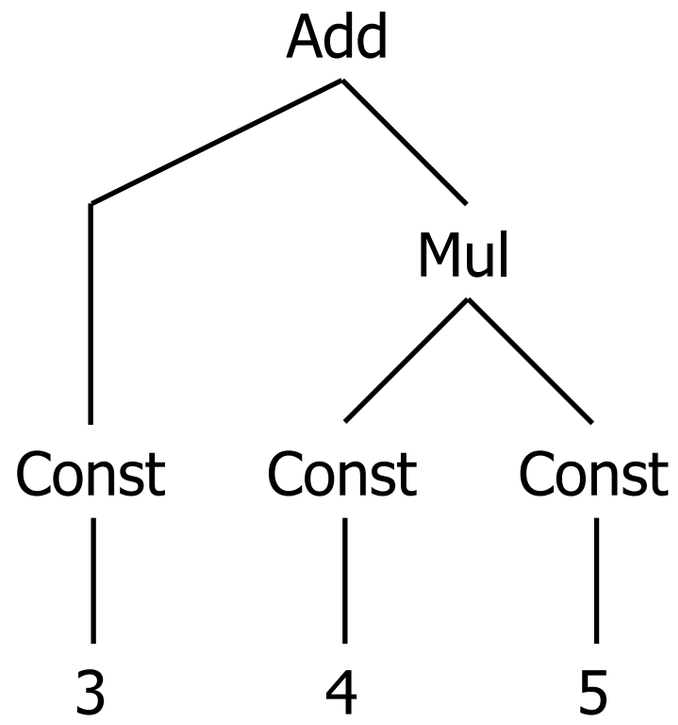
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

topdown(s) = s ; all(topdown(s))
topdown(try(switch))
```



Stratego

example

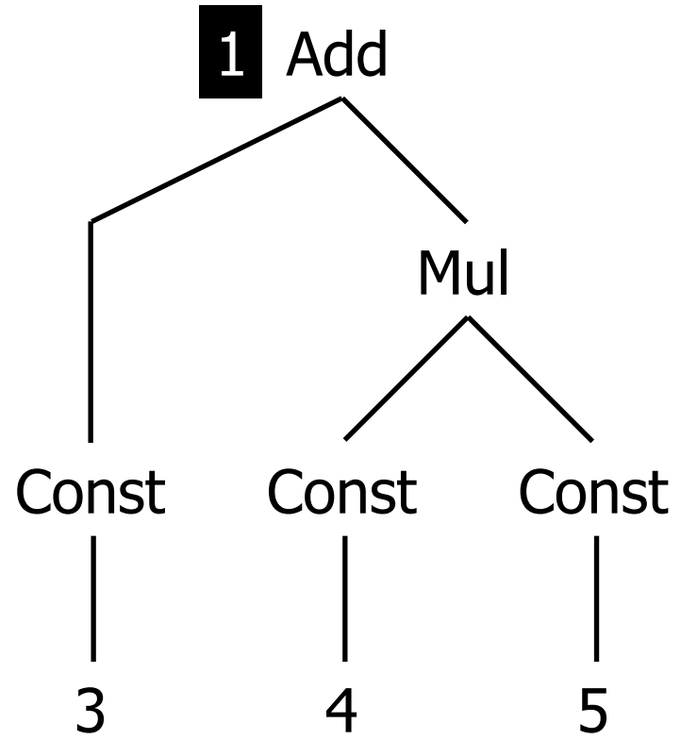
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

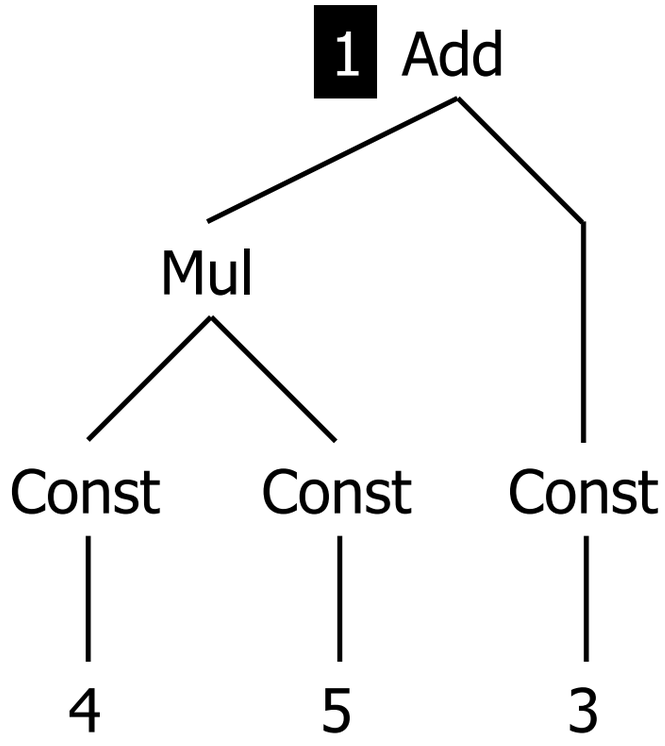
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

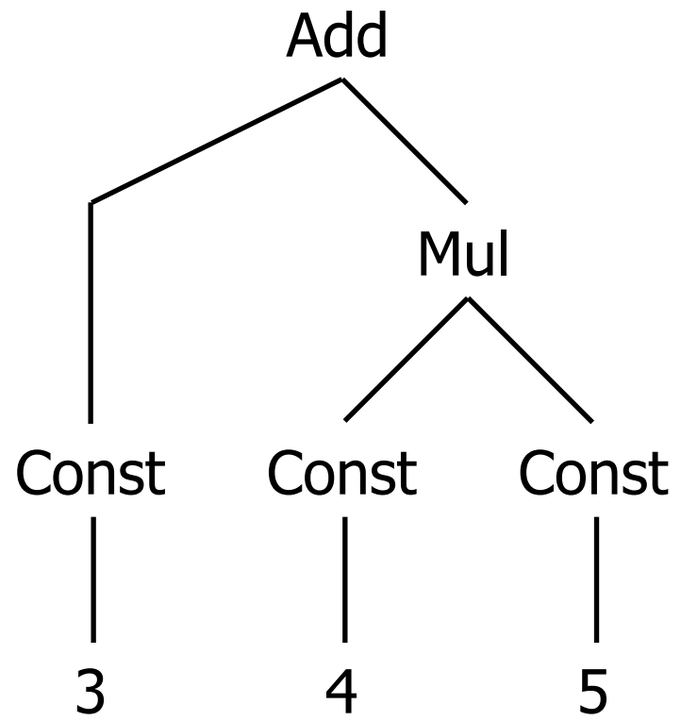
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

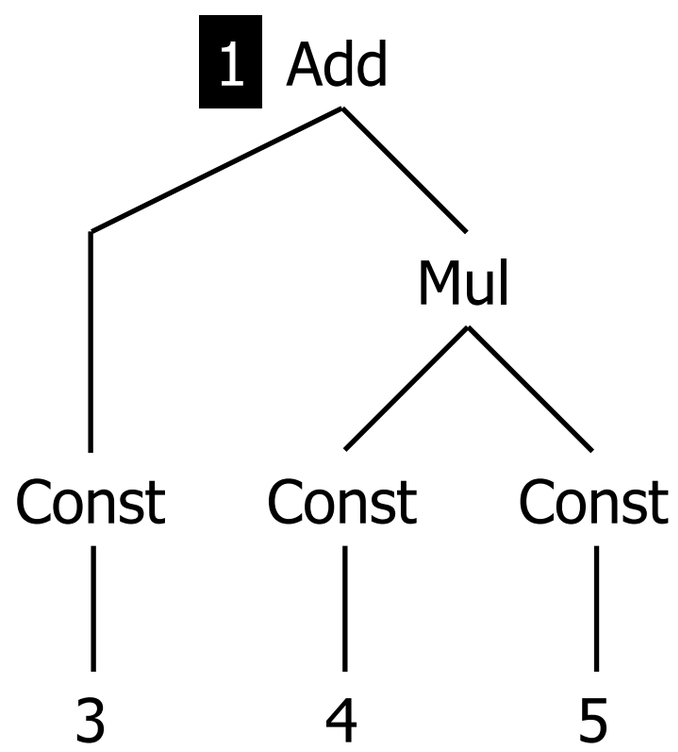
```
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

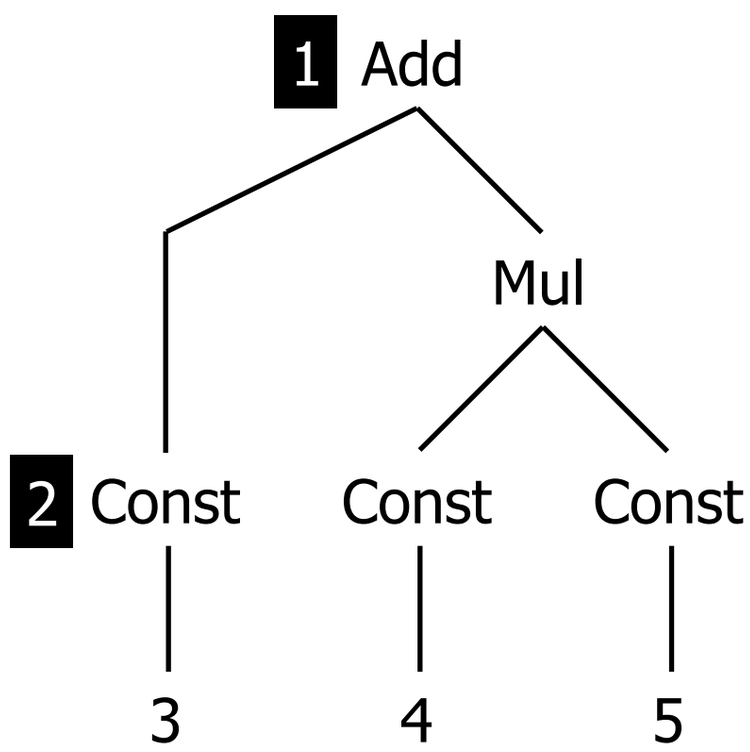
```
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

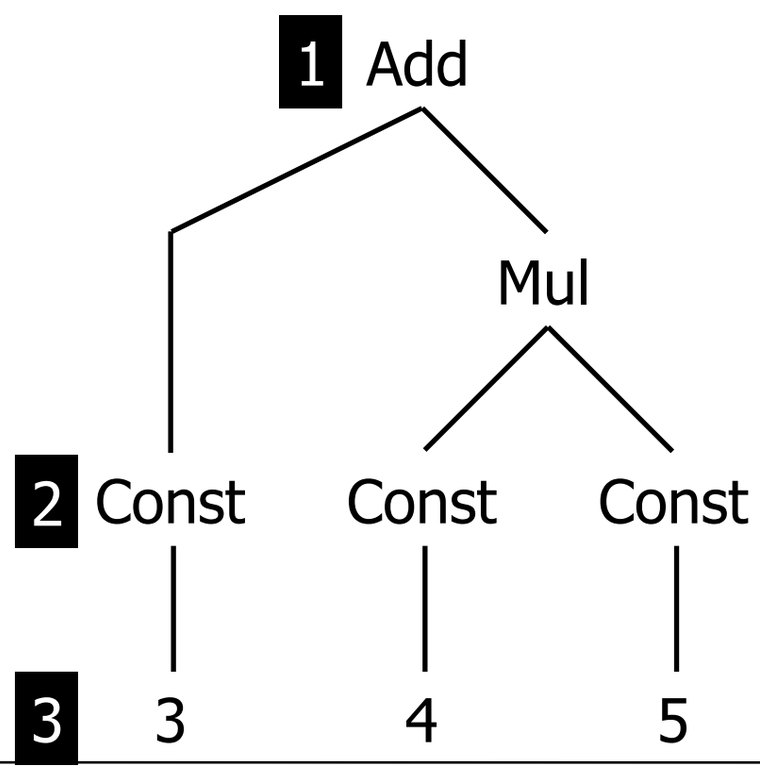
```
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

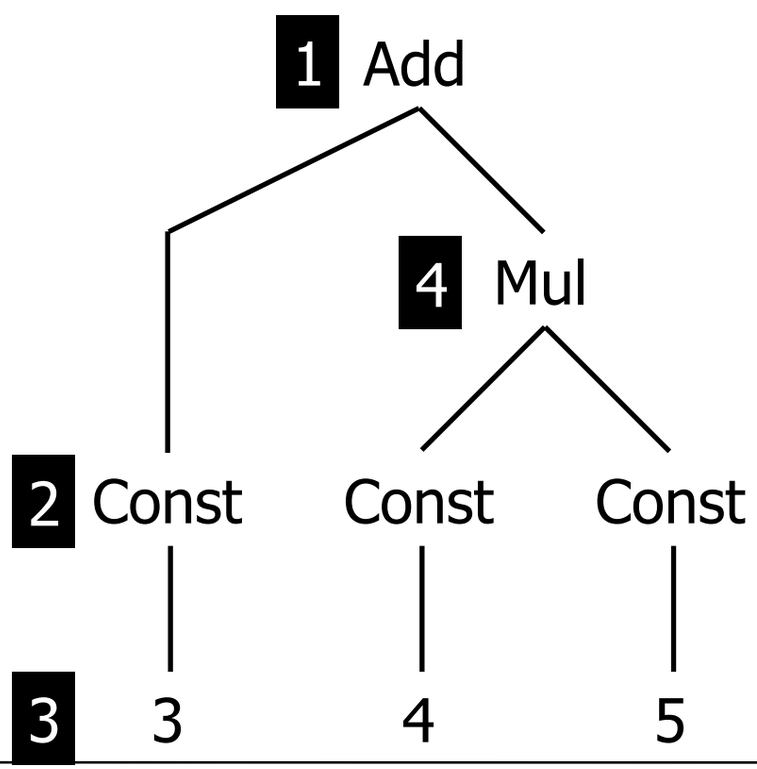
```
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

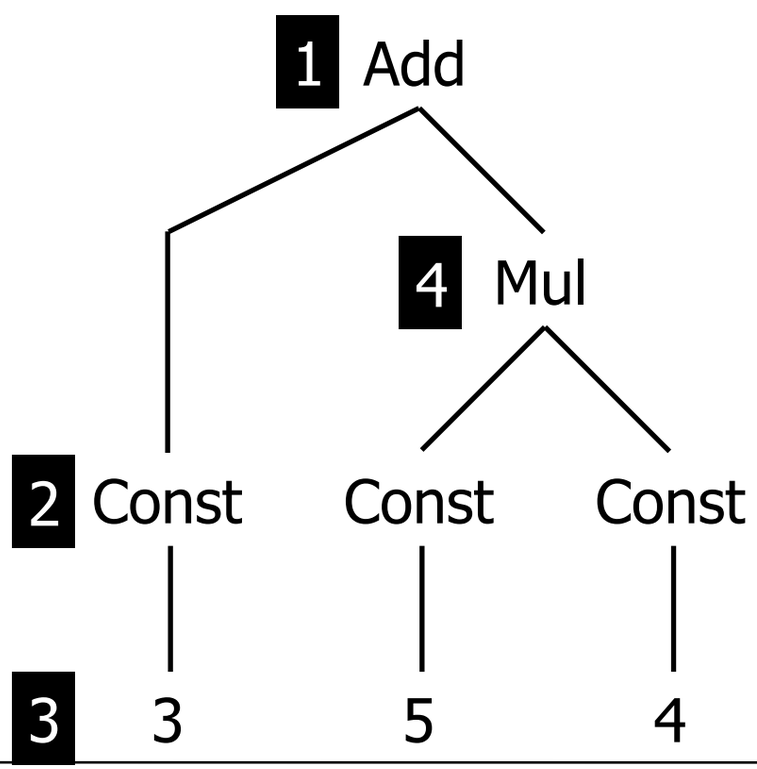
```
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

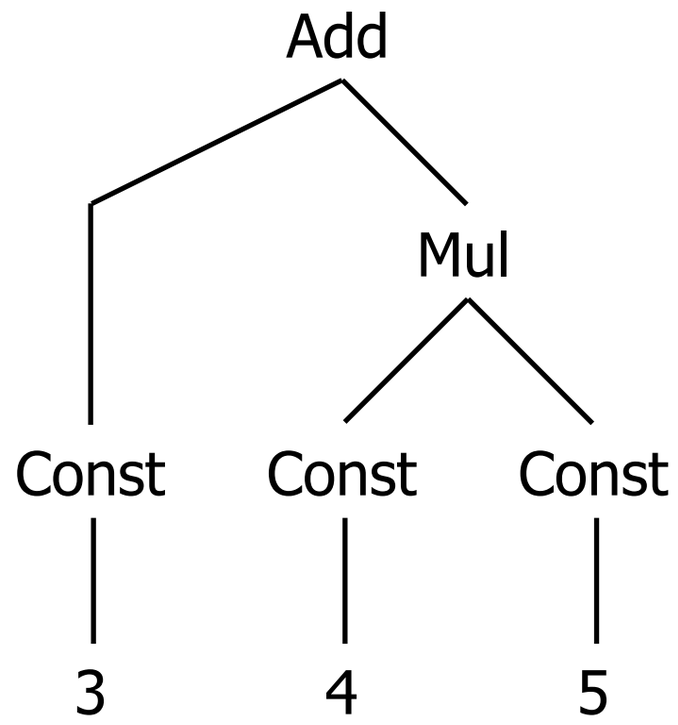
```
switch: Mul(e1, e2) -> Mul(e2, e1)

alltd(s) = s <+ all(alltd(s))

alltd(switch)
```



Stratego

example

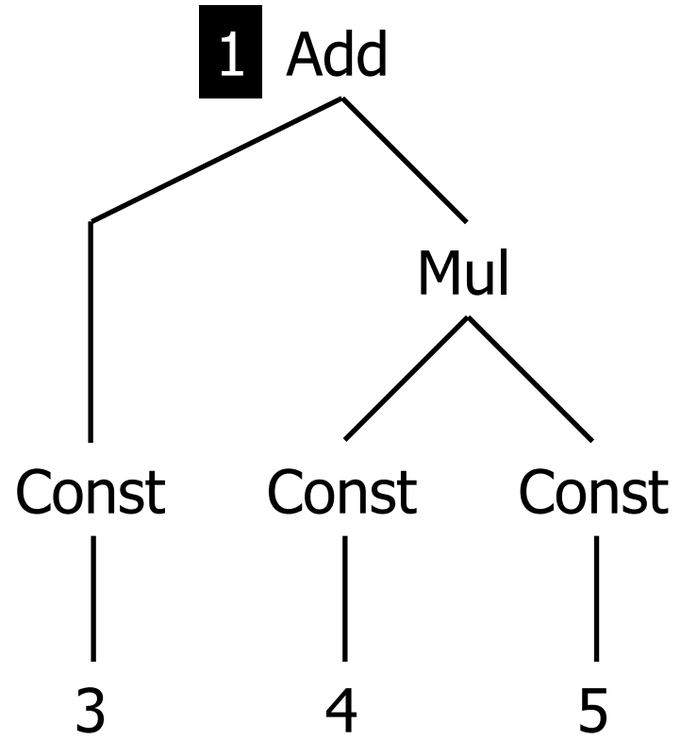
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s

bottomup(switch)
```



Stratego

example

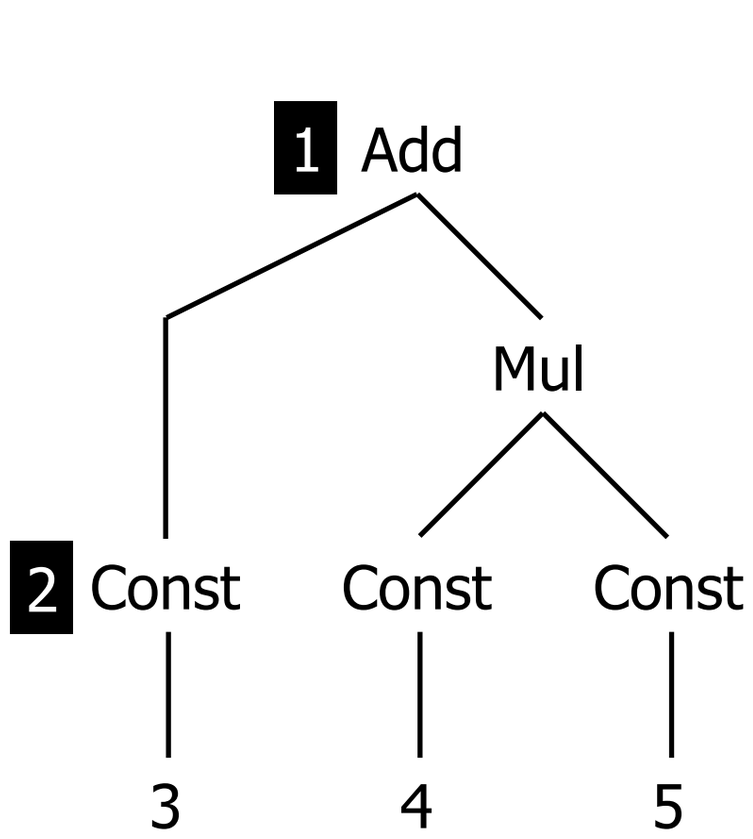
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s

bottomup(switch)
```



Stratego

example

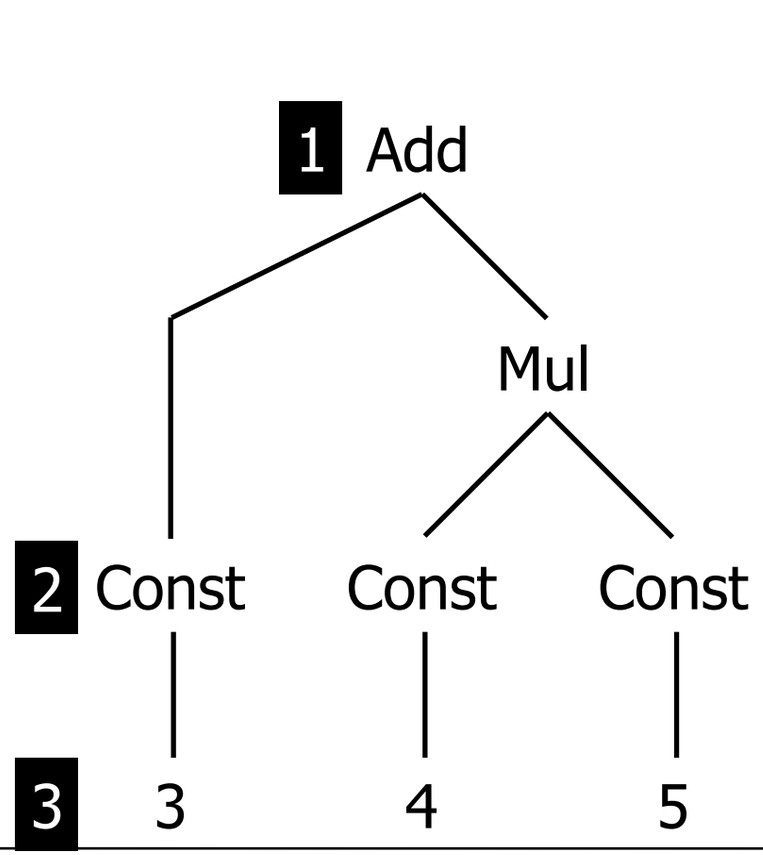
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(switch)
```



Stratego

example

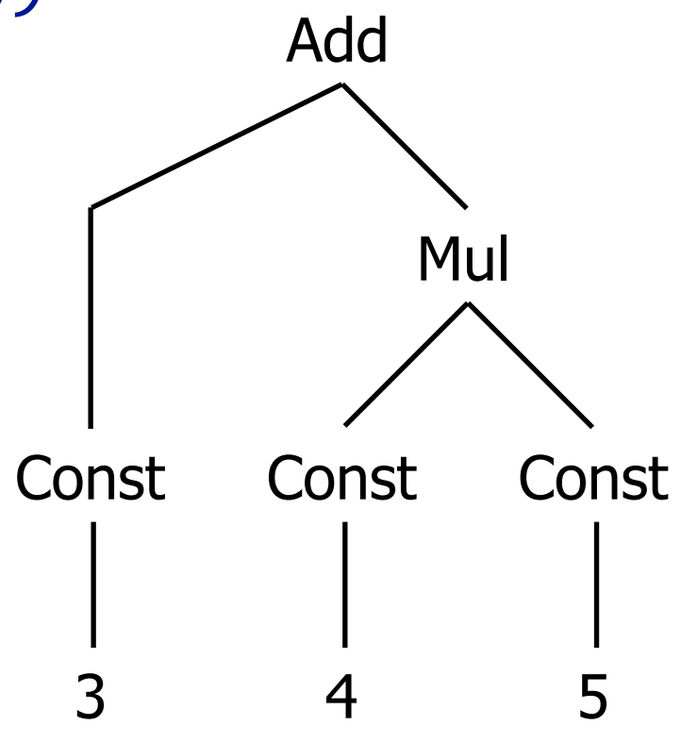
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(switch)
```



Stratego

example

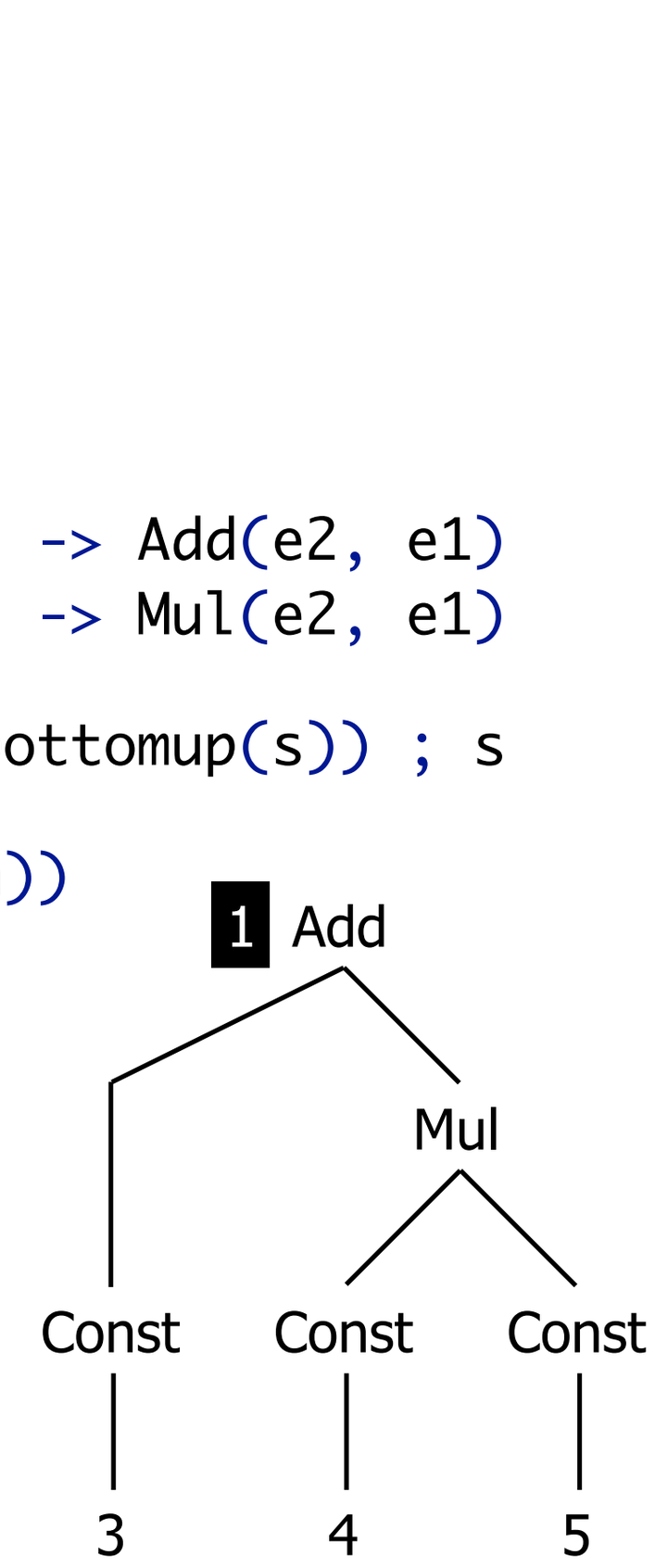
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

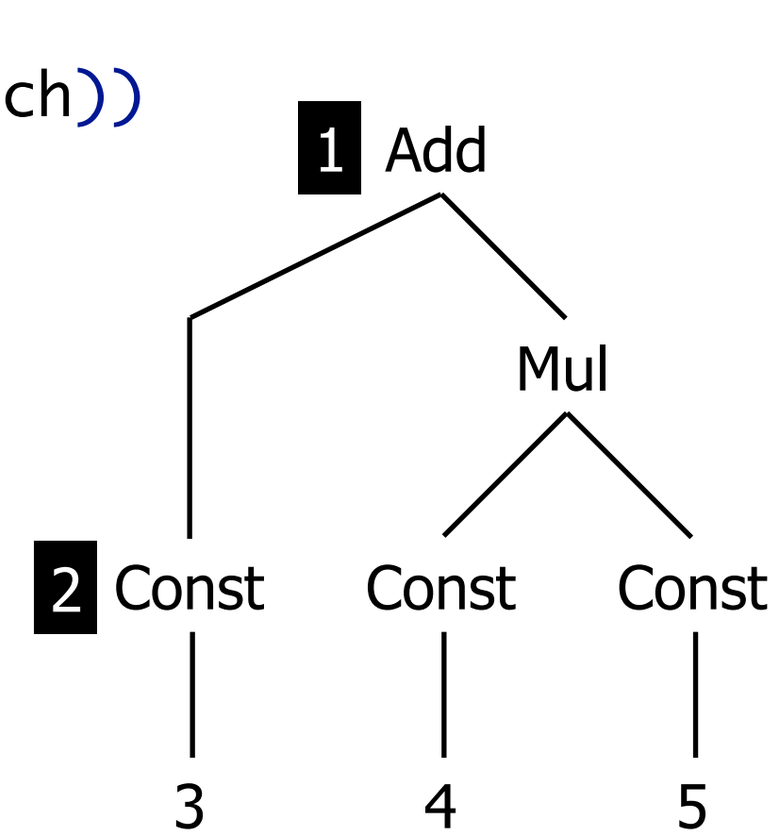
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

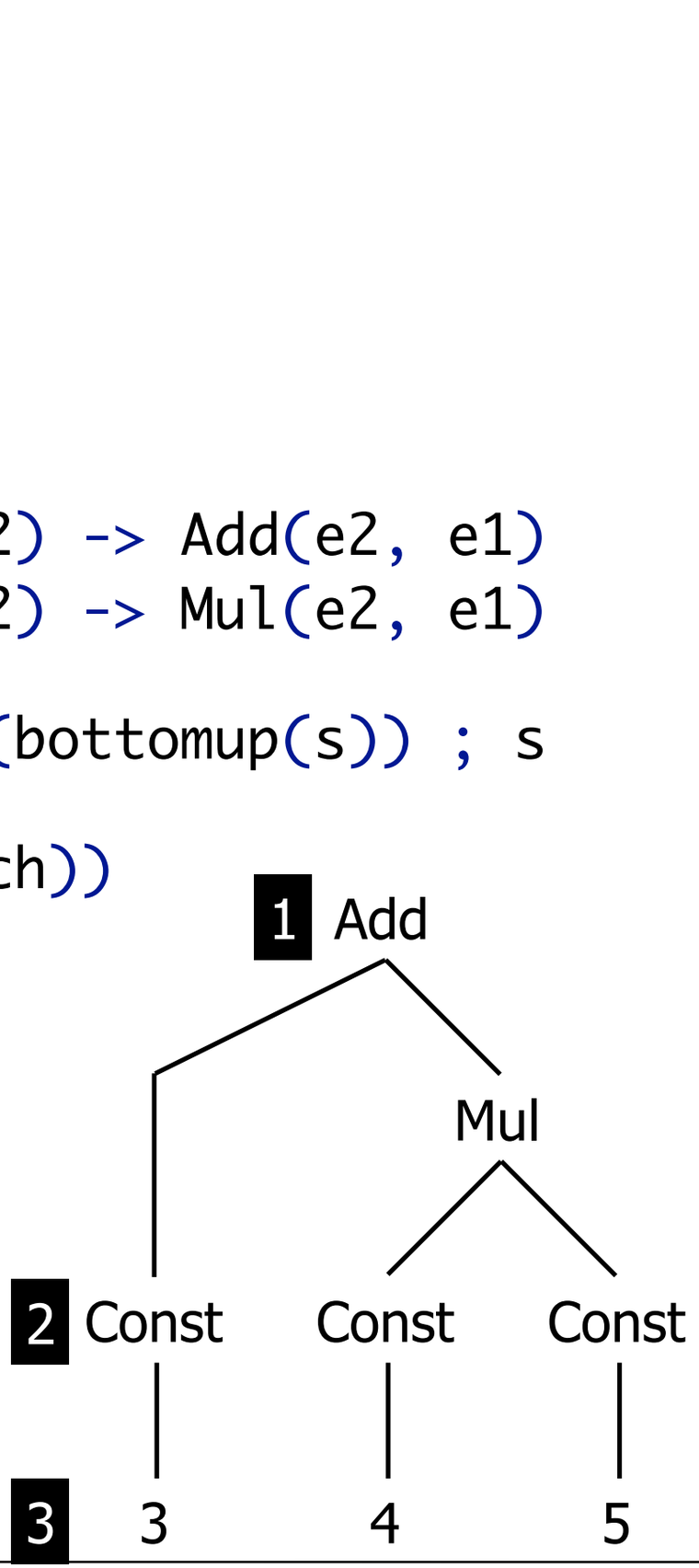
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

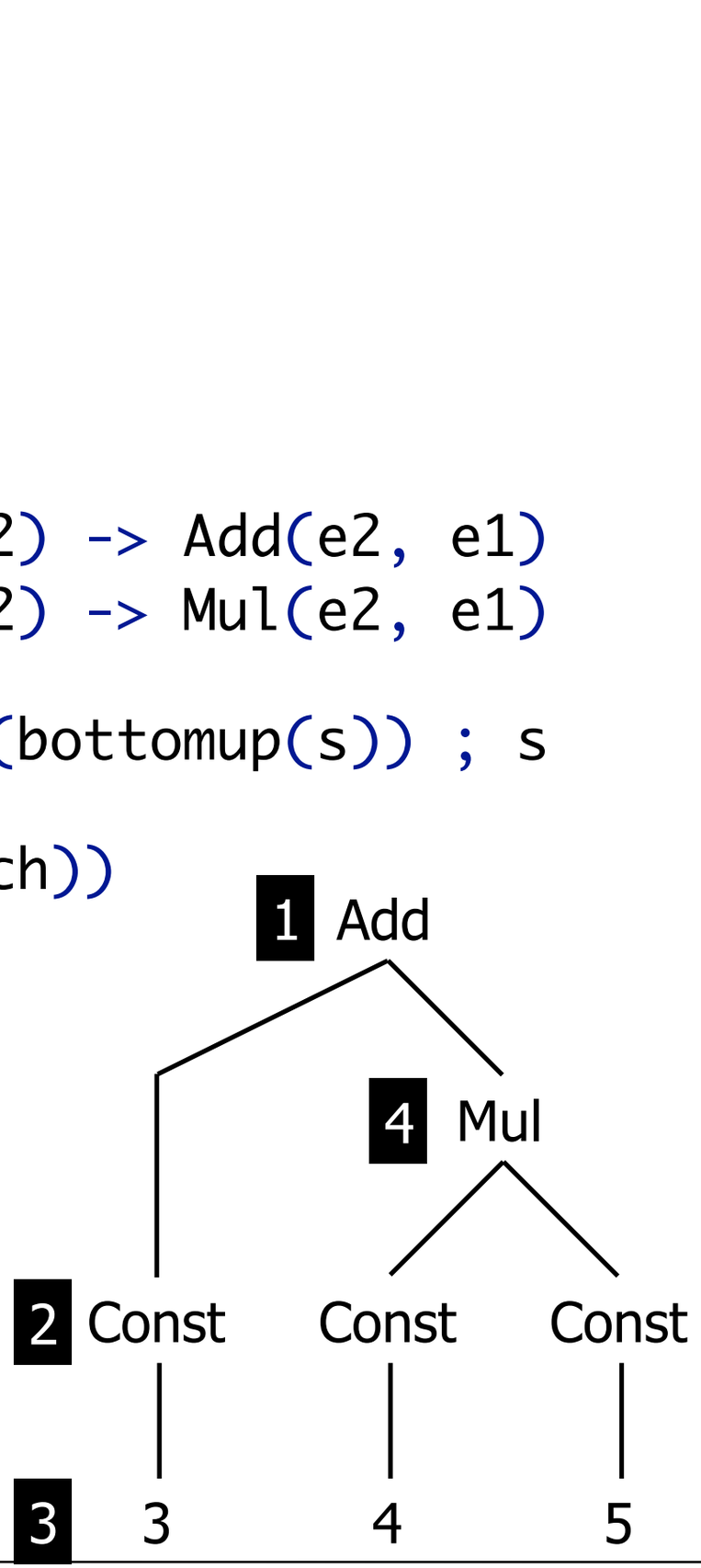
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

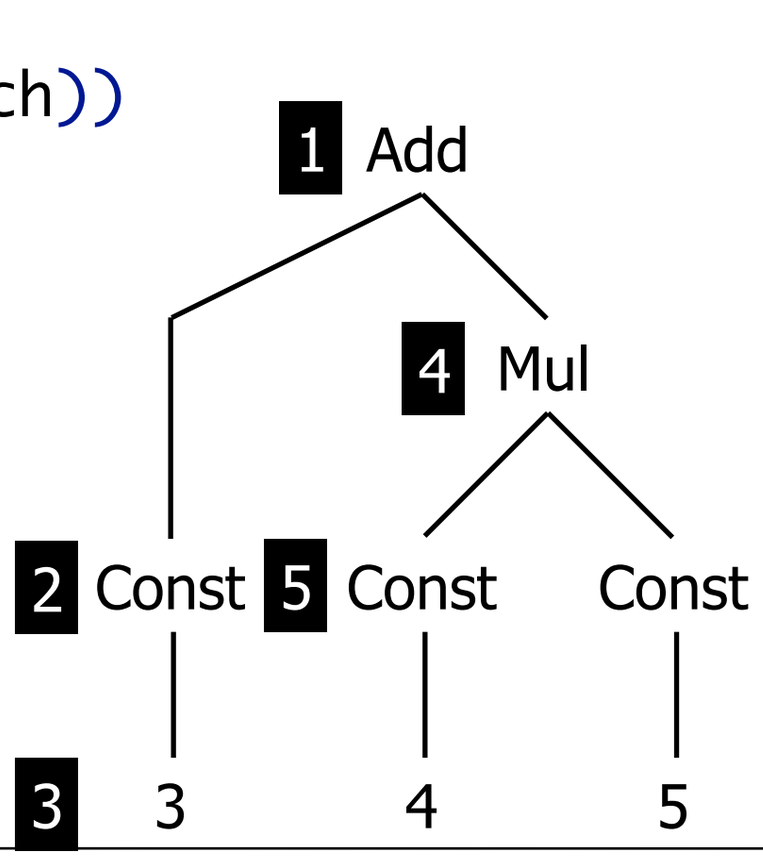
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

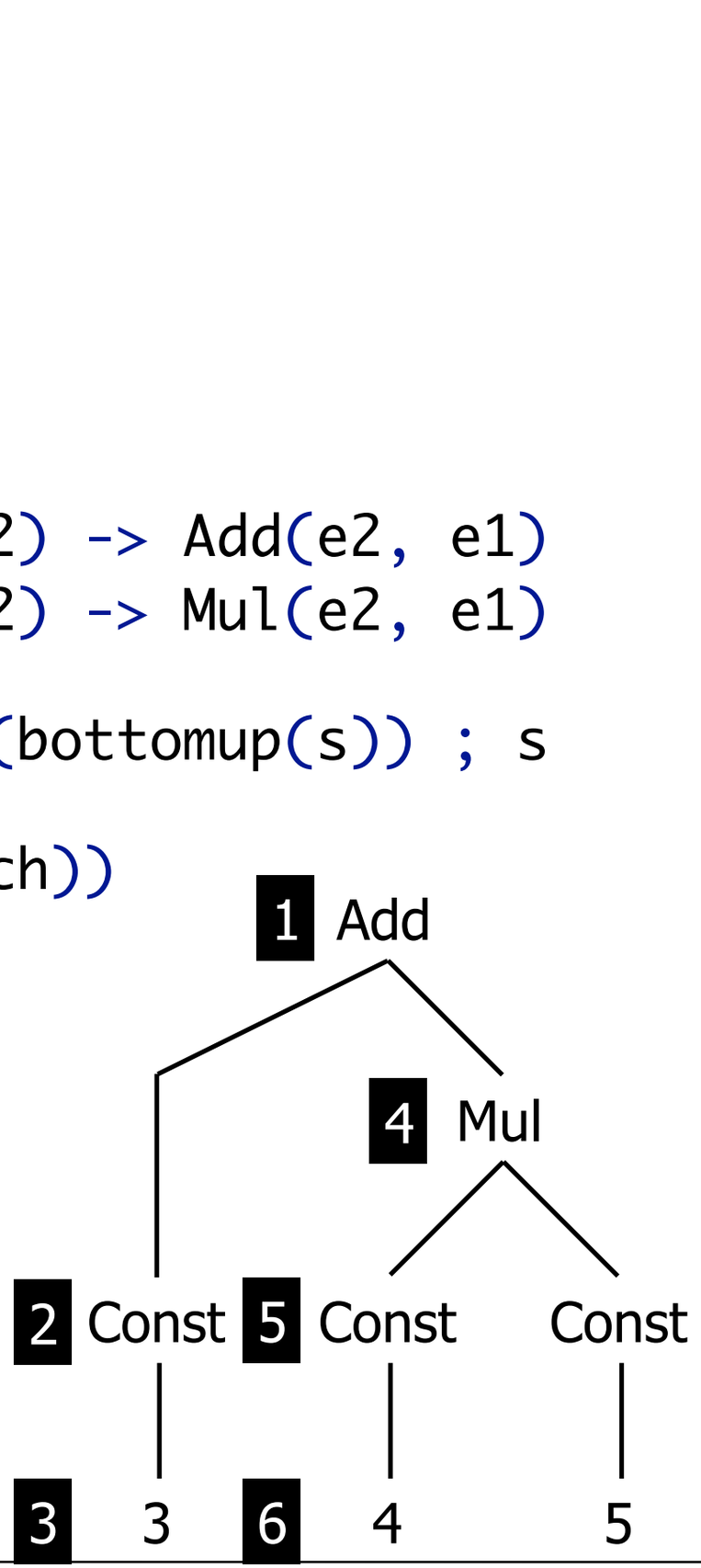
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

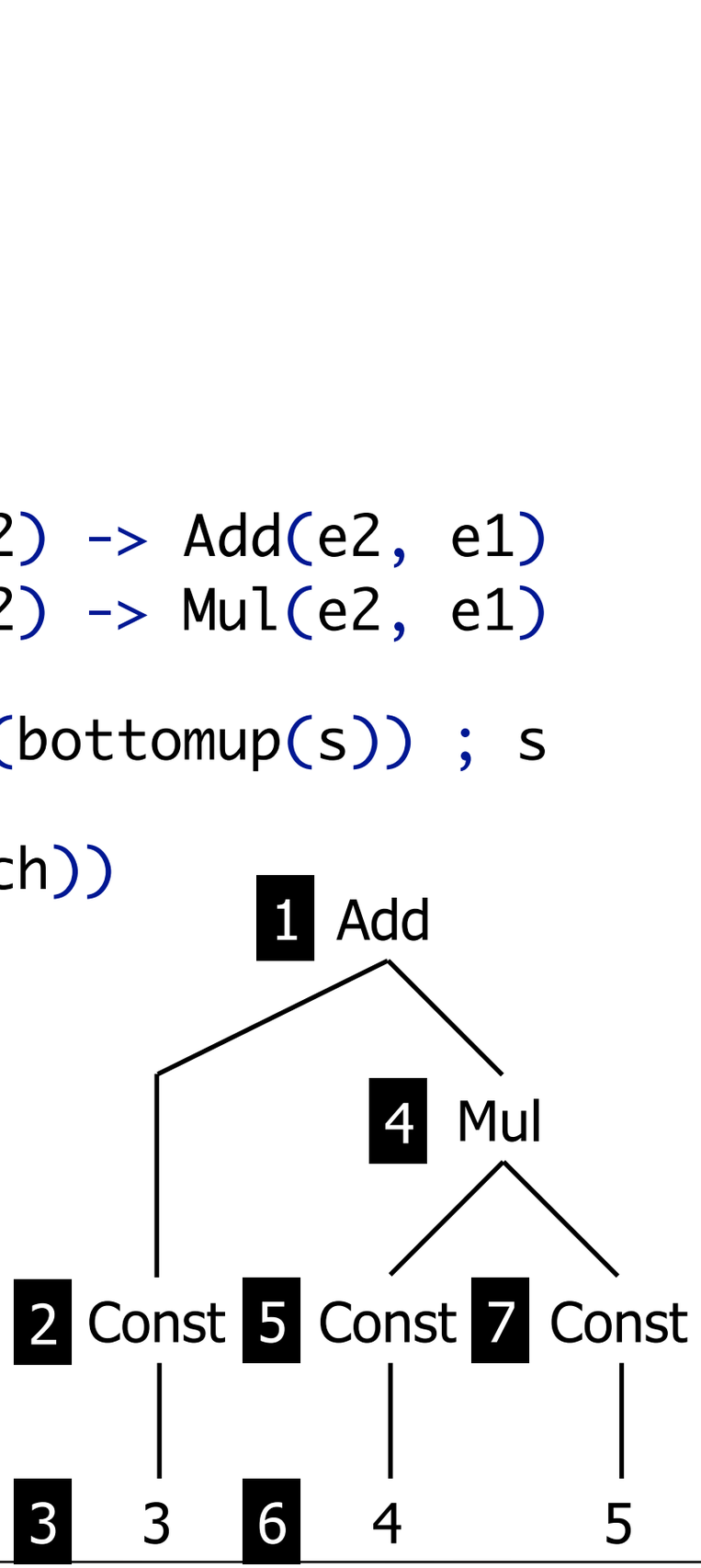
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

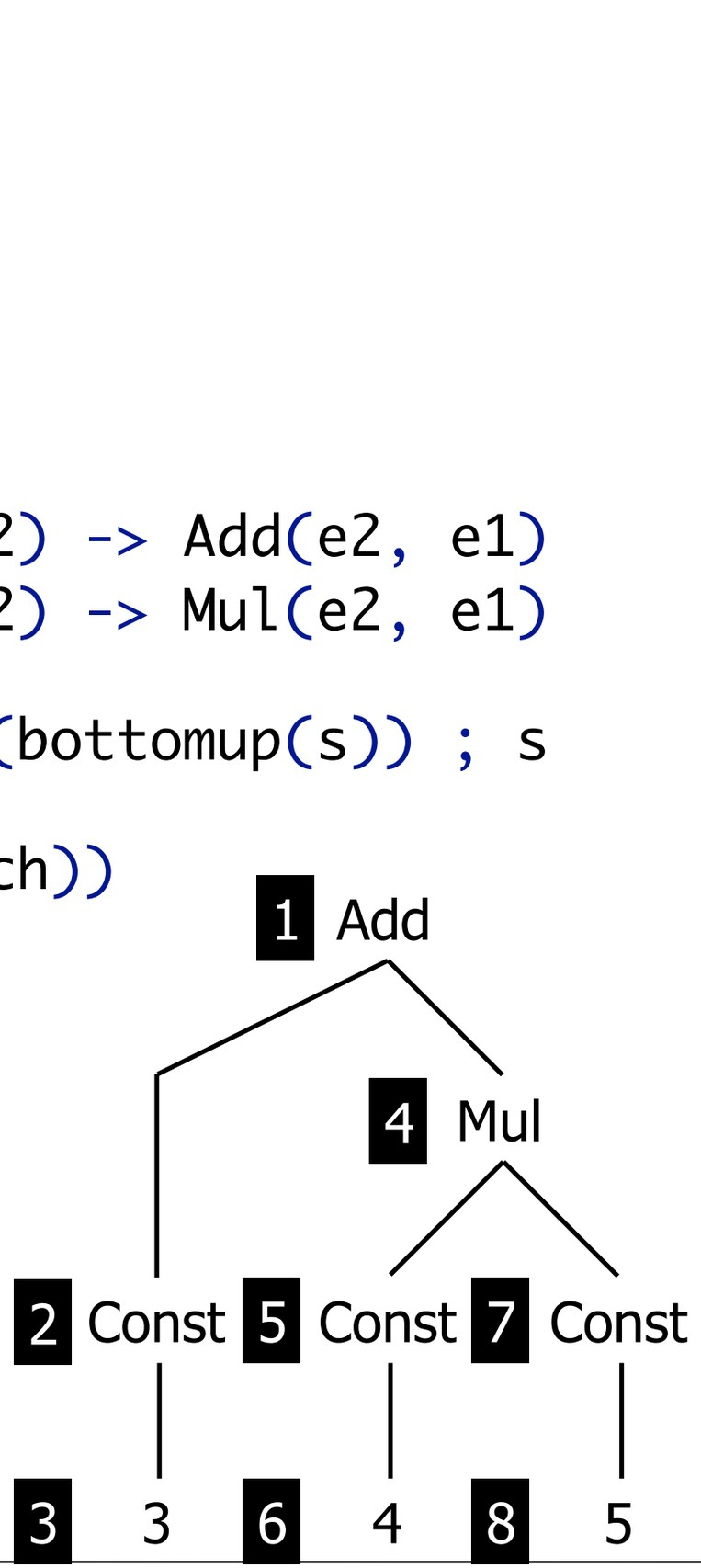
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

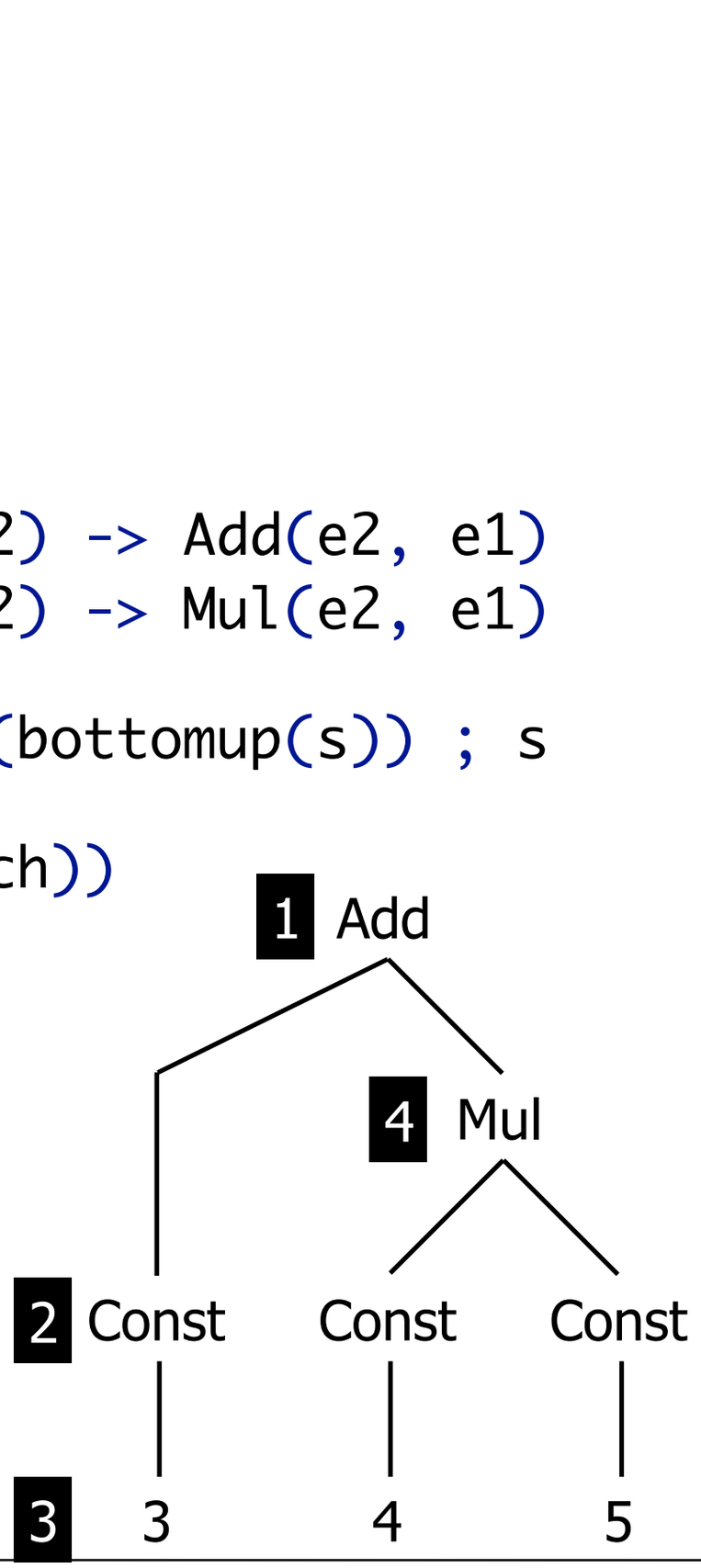
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

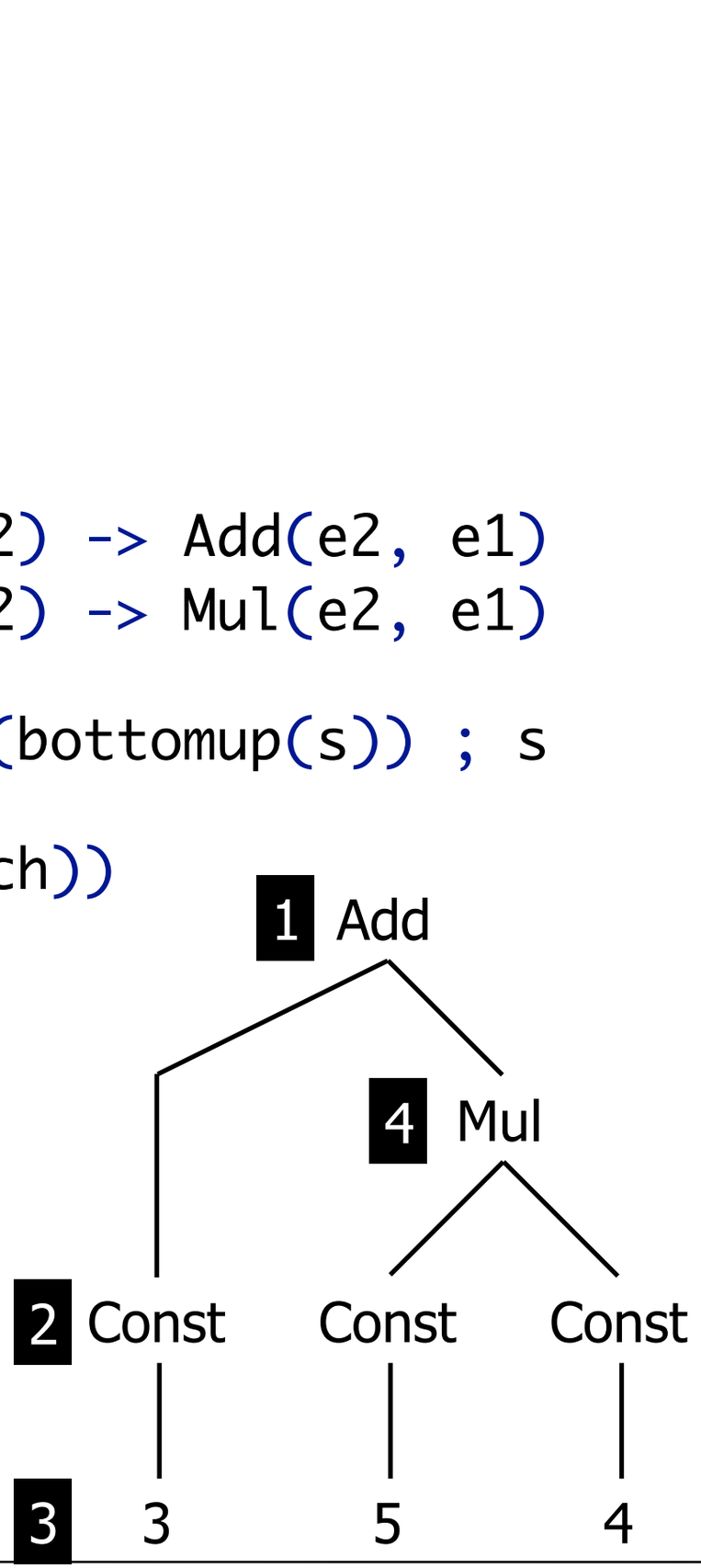
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

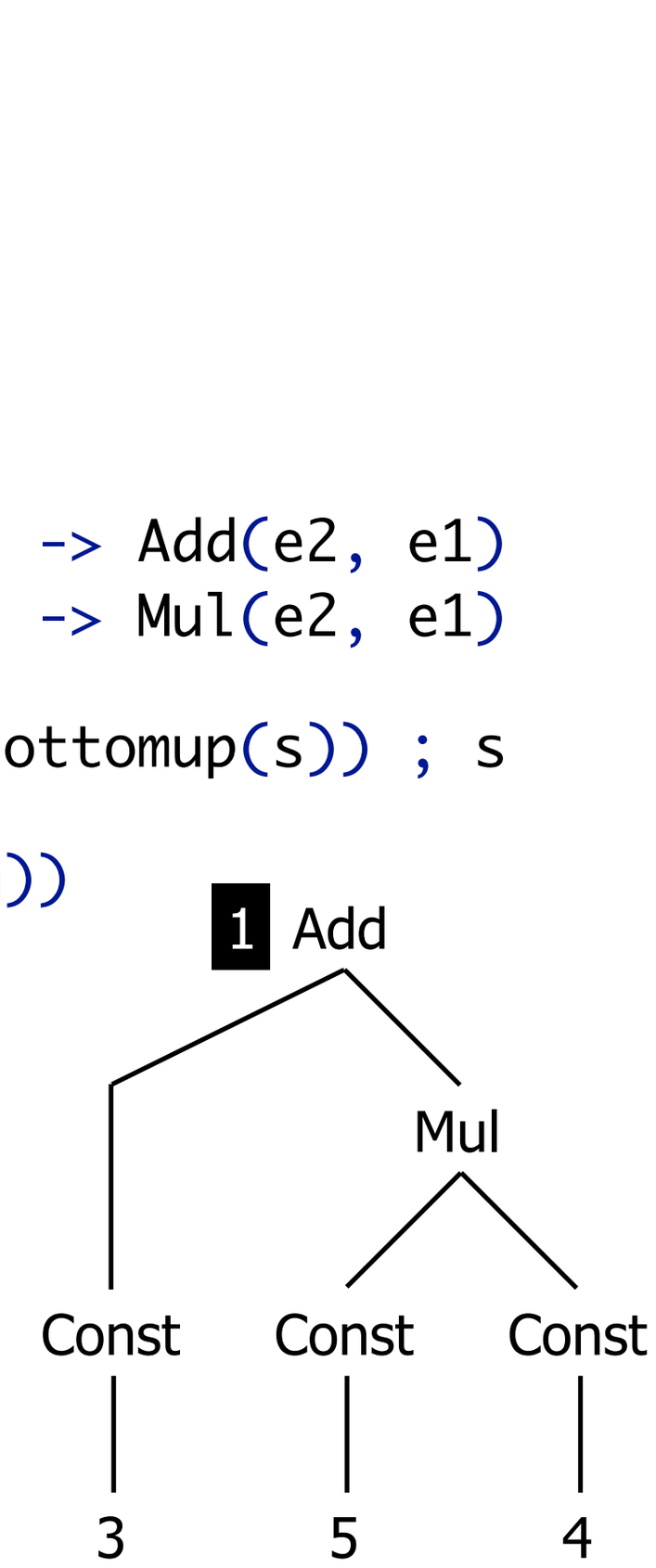
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

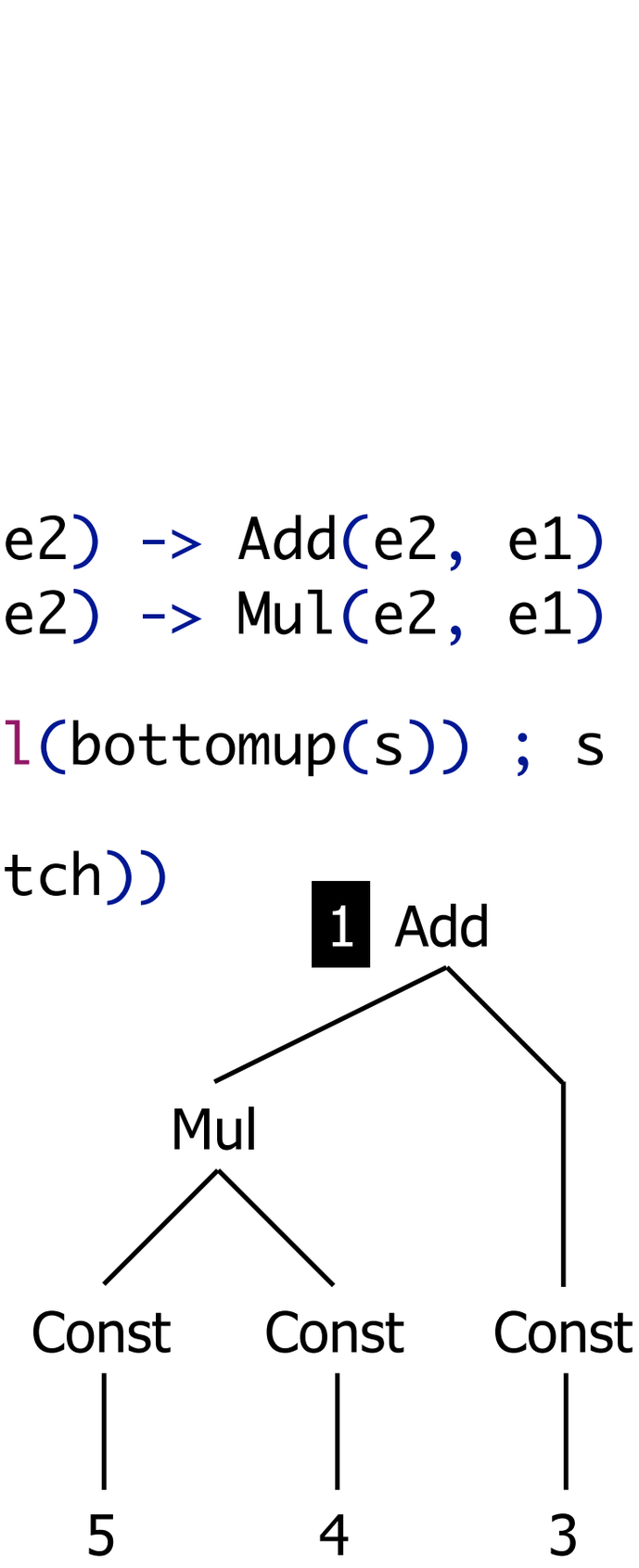
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

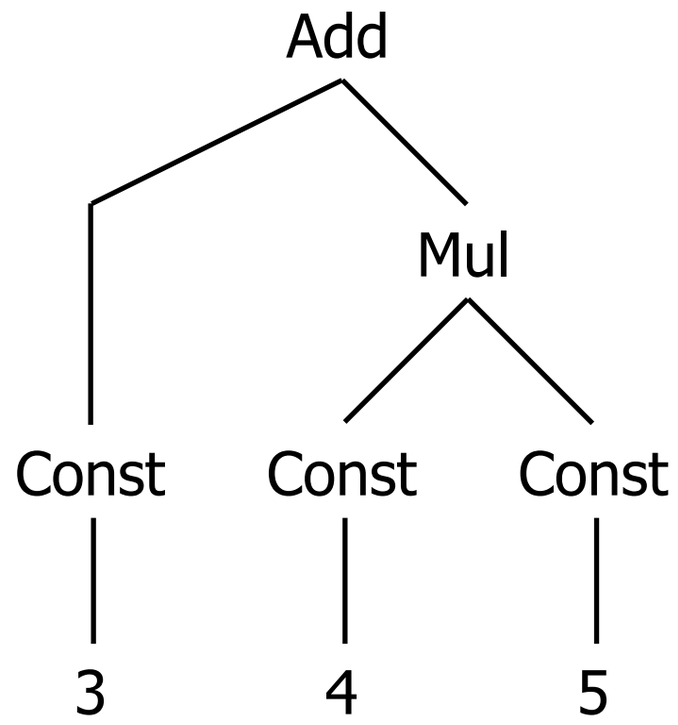
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

bottomup(s) = all(bottomup(s)) ; s
bottomup(try(switch))
```



Stratego

example

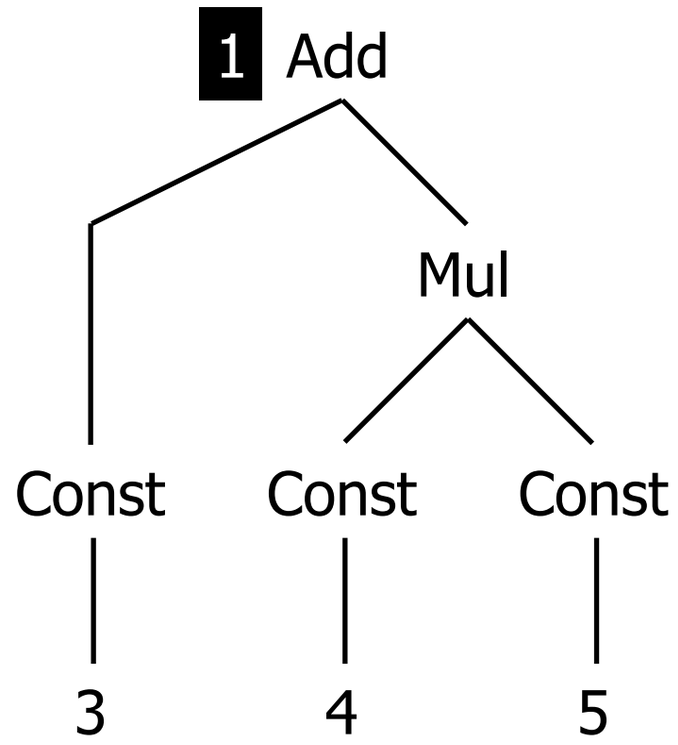
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

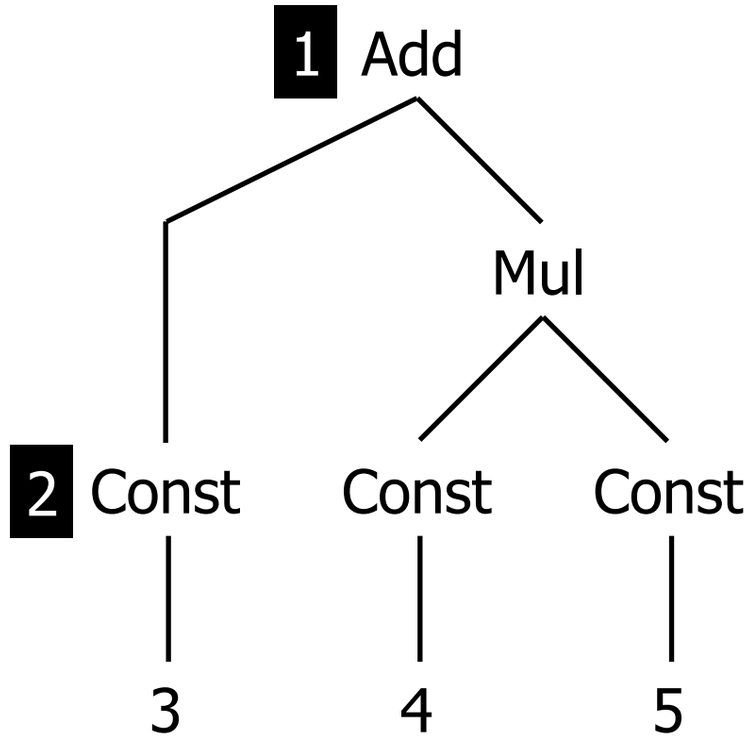
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

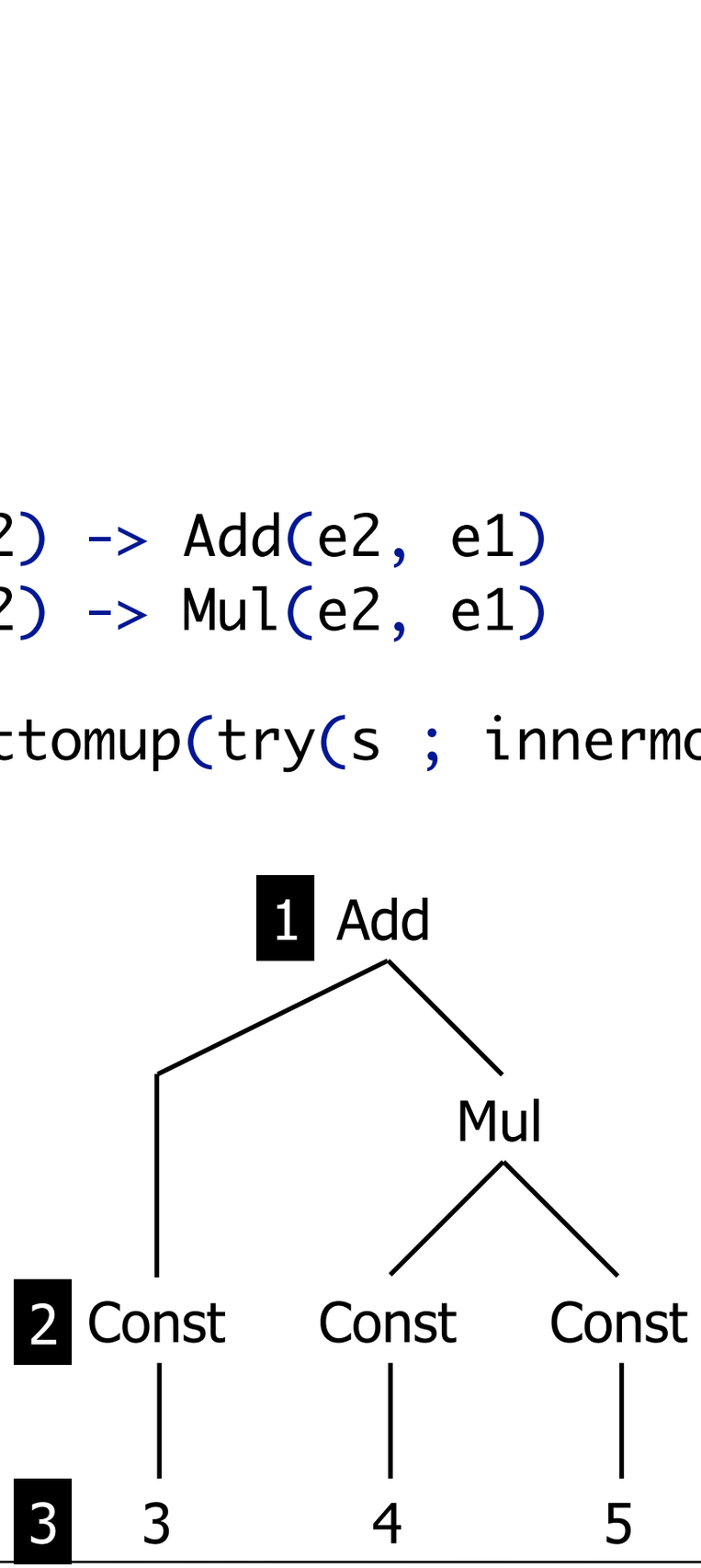
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

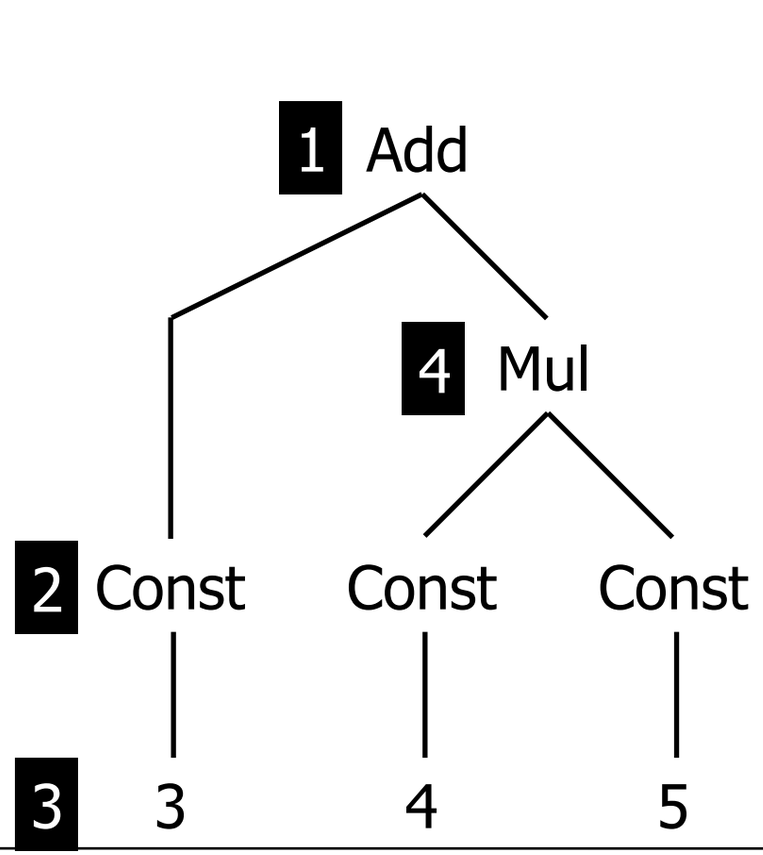
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

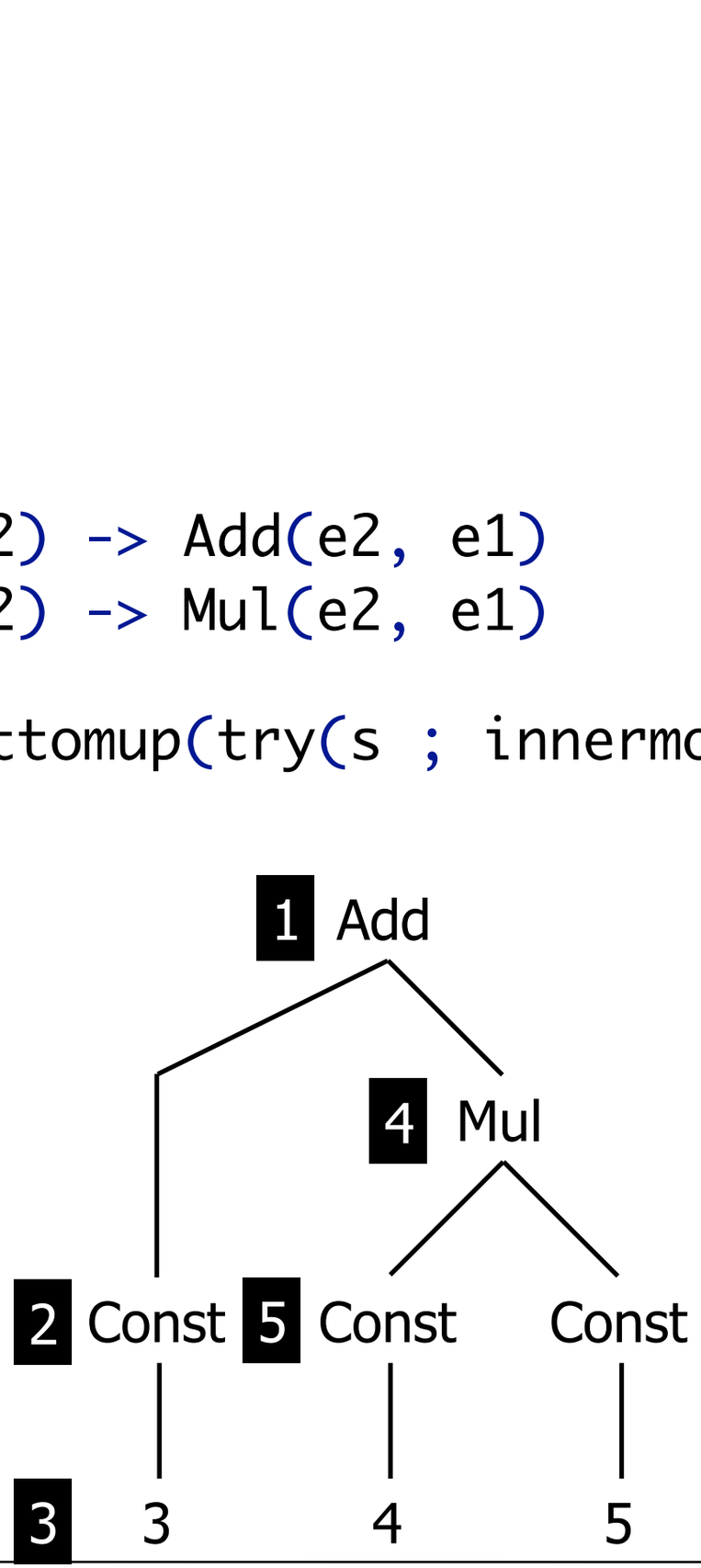
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

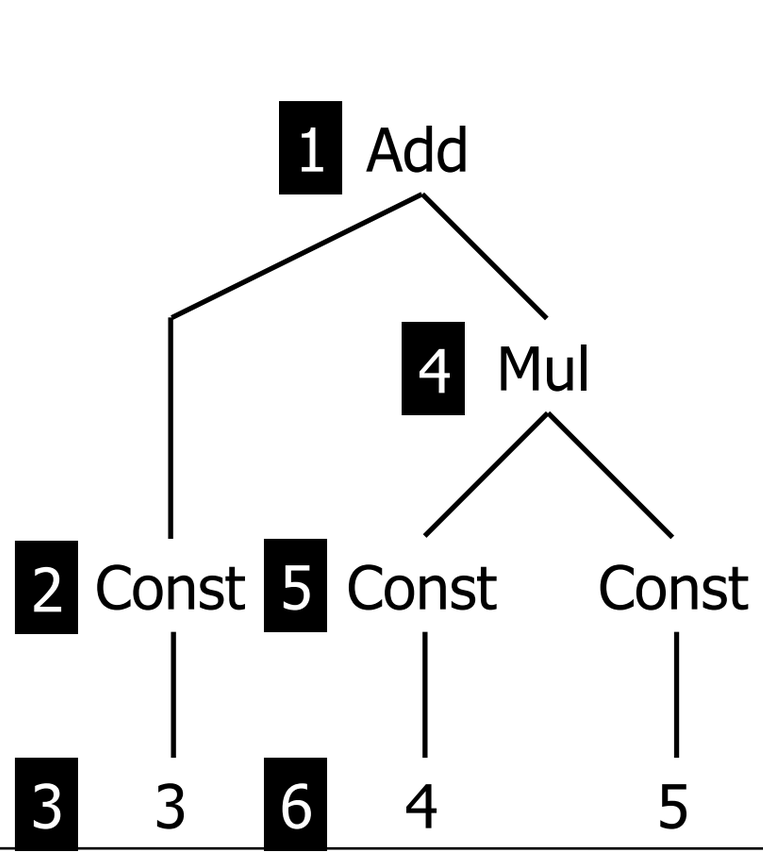
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

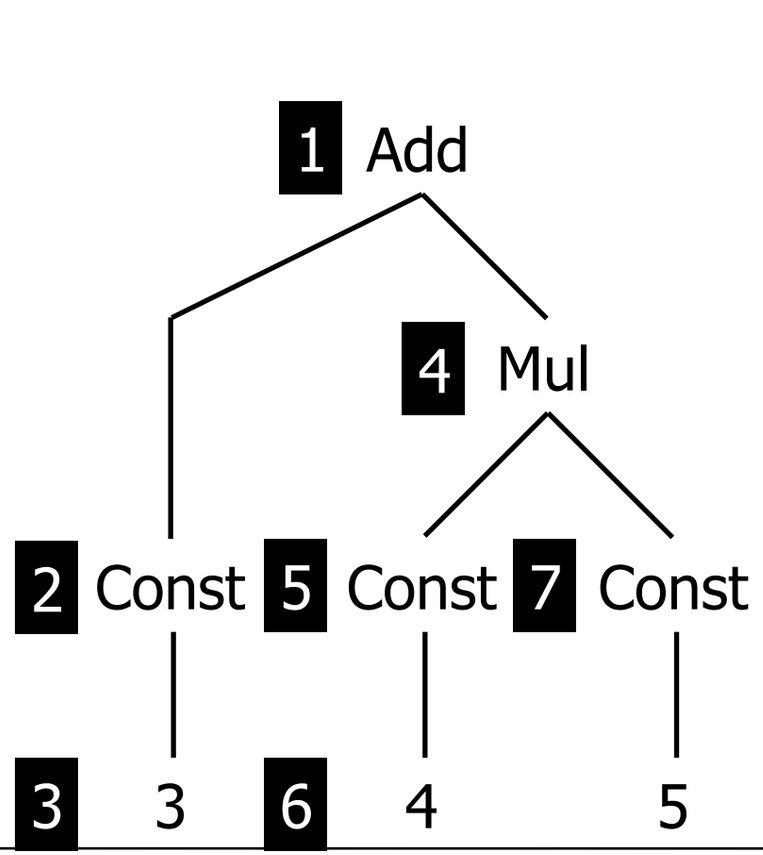
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

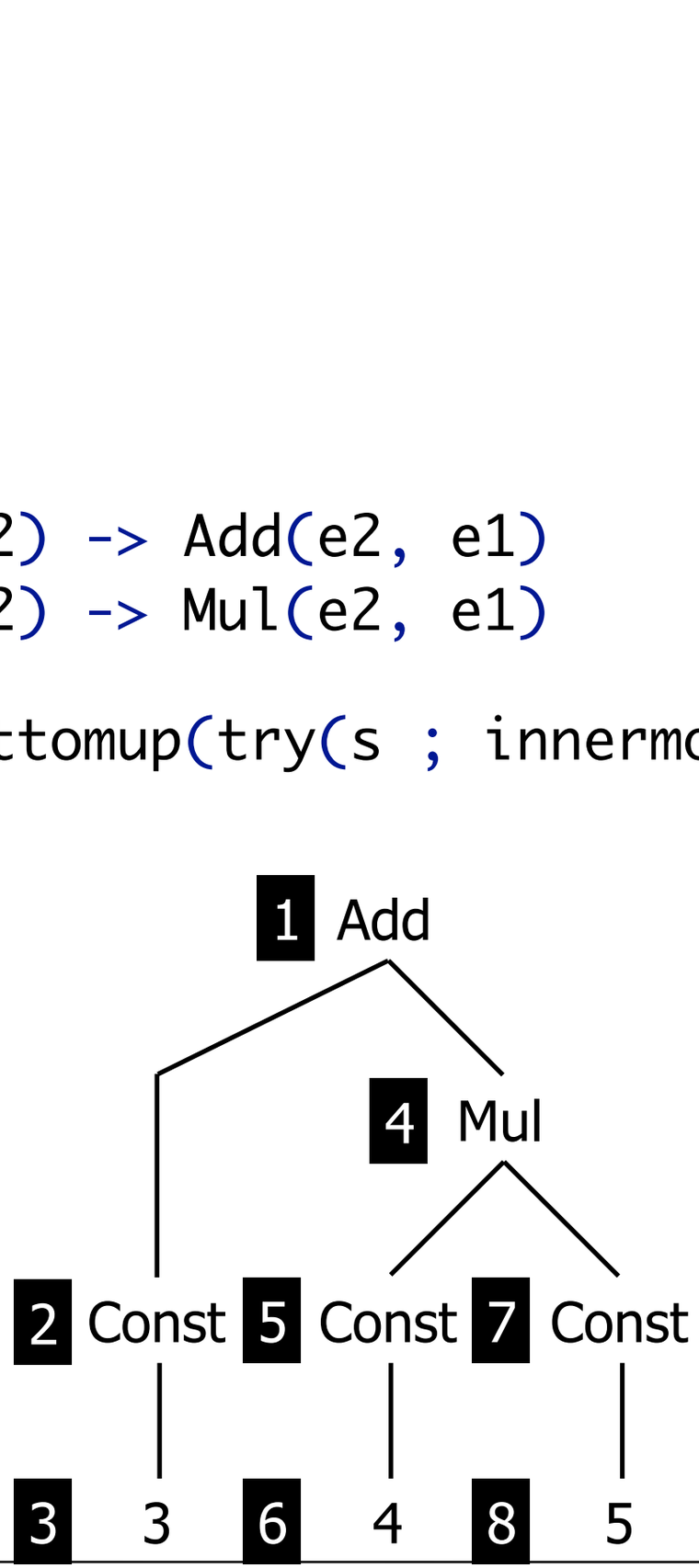
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

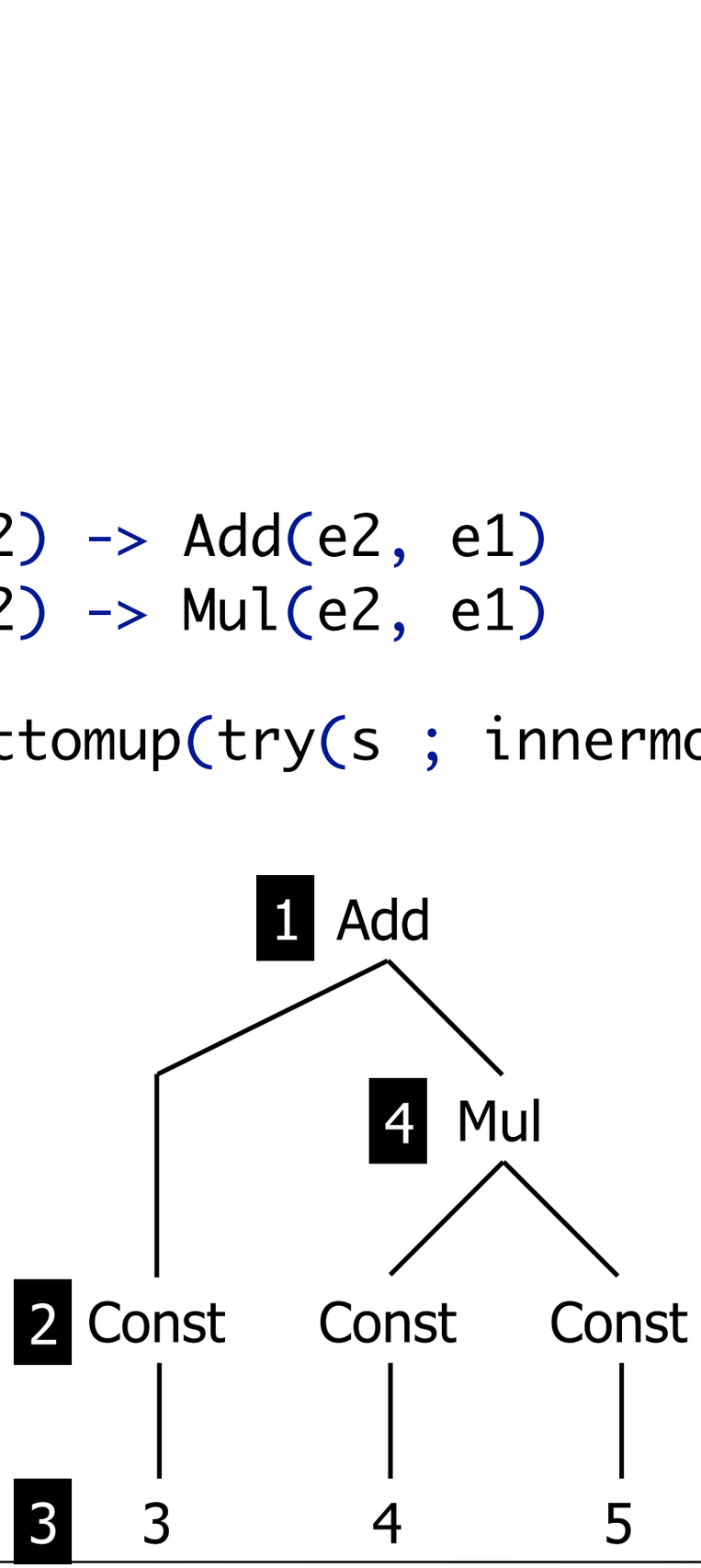
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

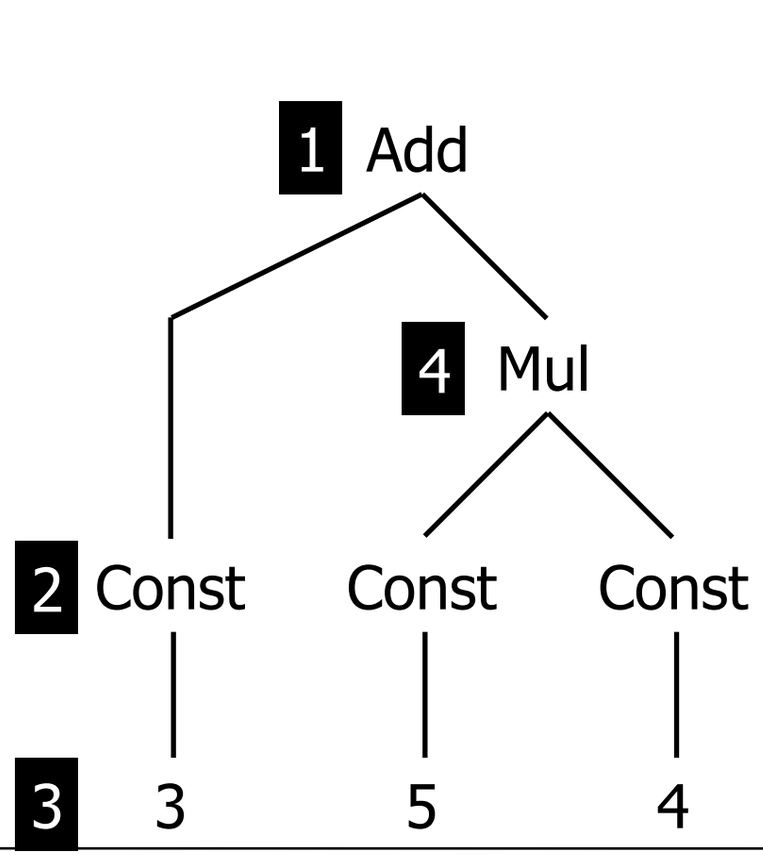
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

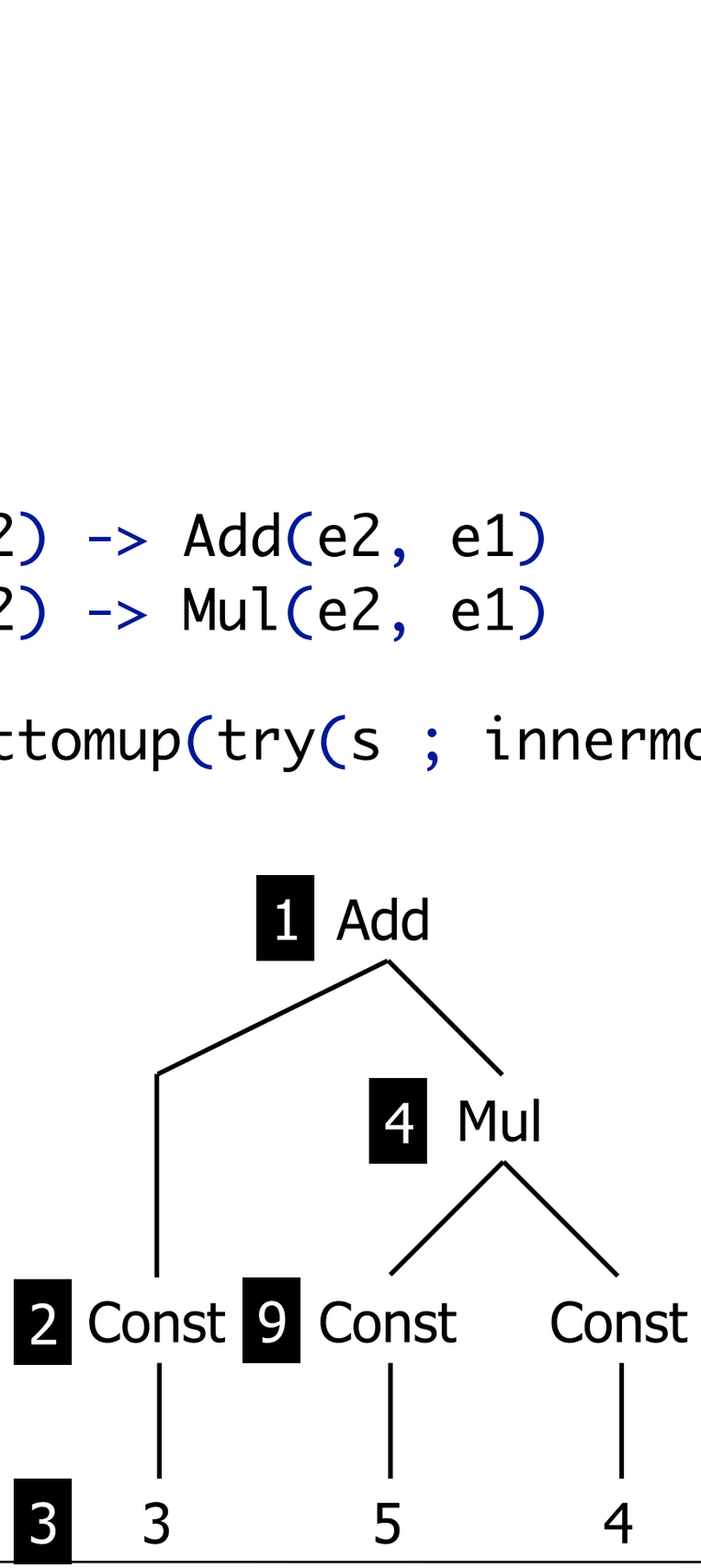
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

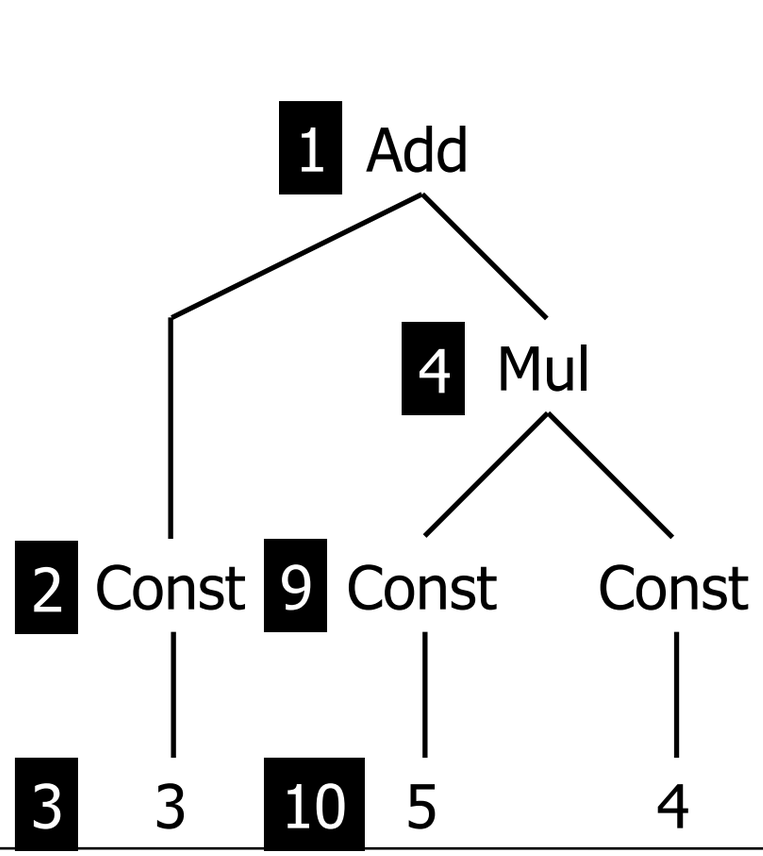
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

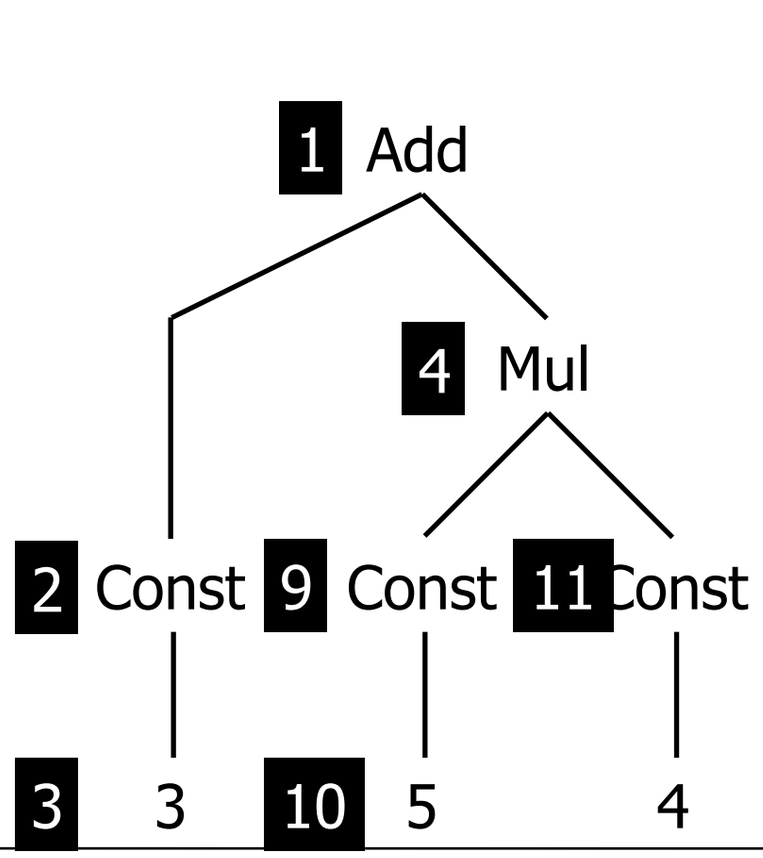
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

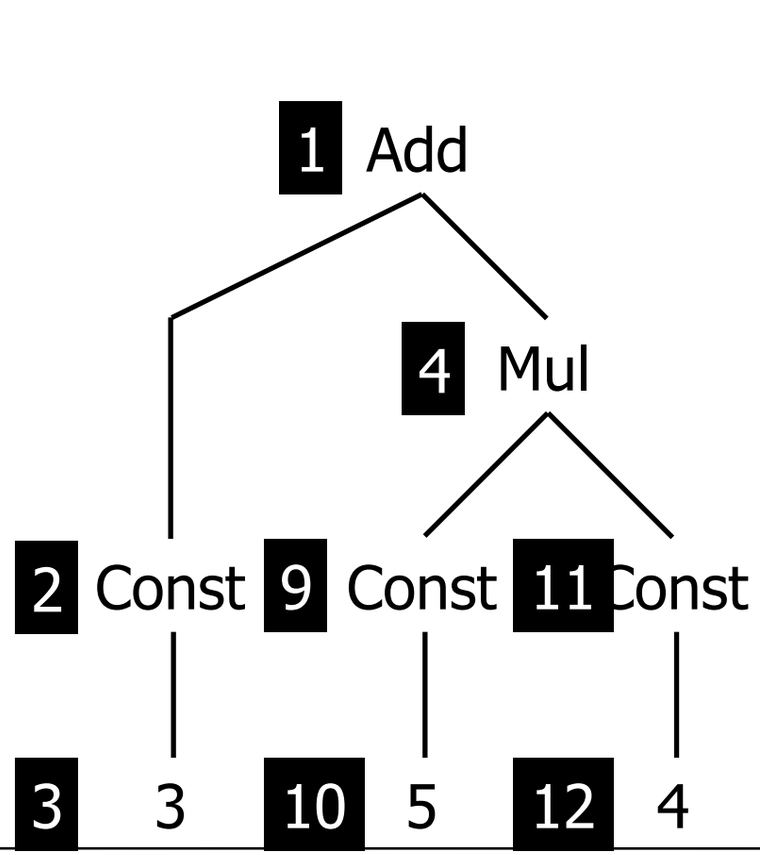
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

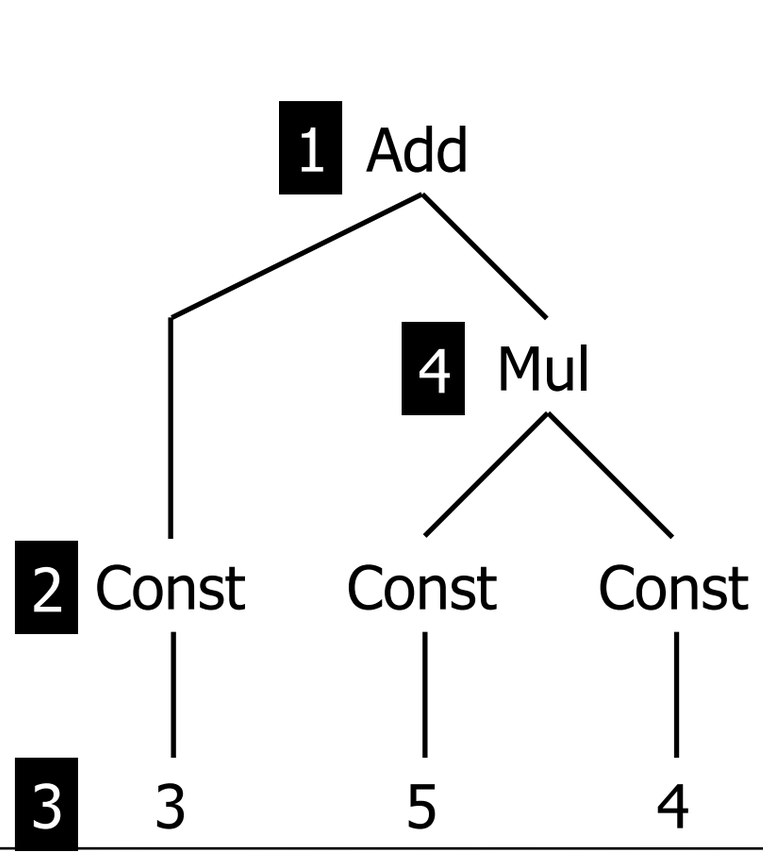
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

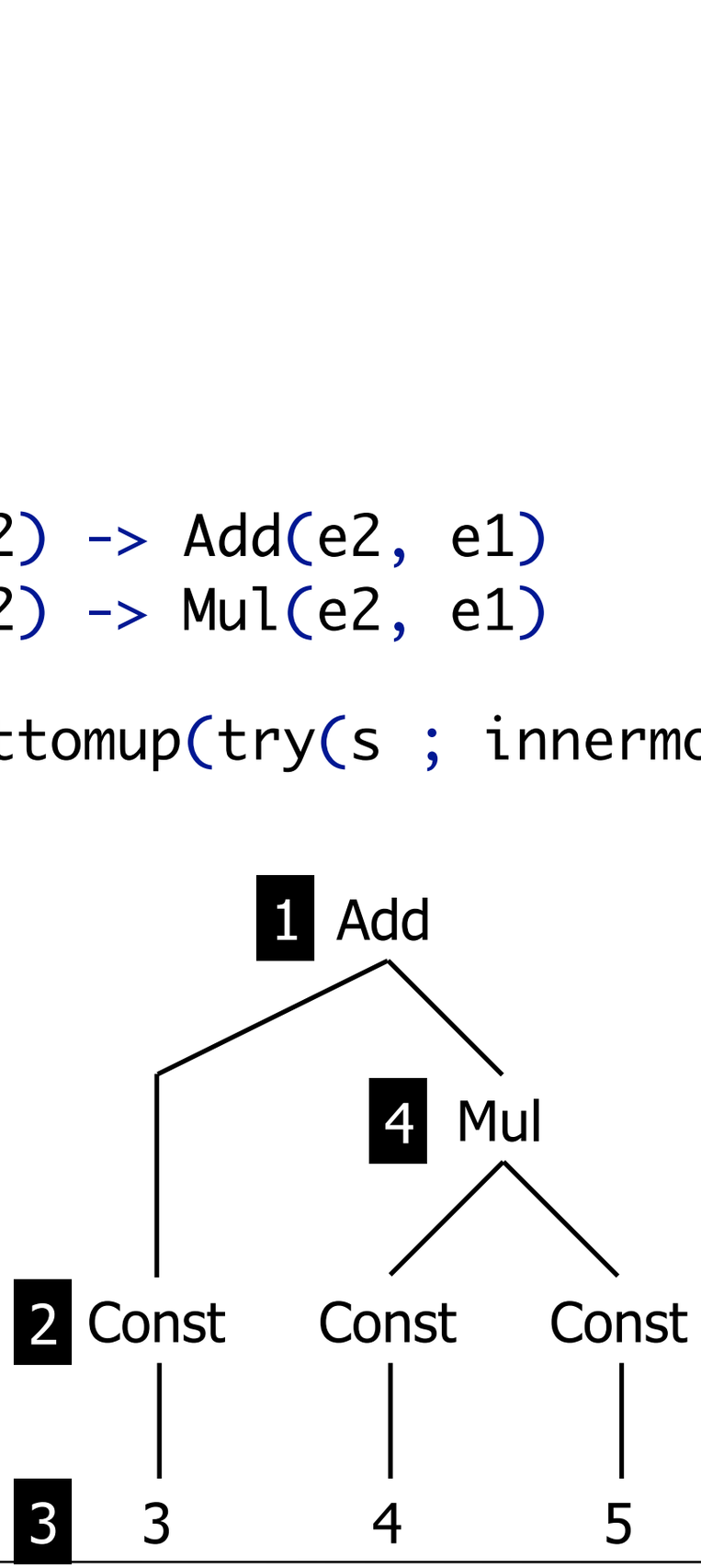
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



Stratego

example

```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

innermost(s) = bottomup(try(s ; innermost(s)))

innermost(switch)
```



VI

summary

Summary
lessons learned

Summary
lessons learned

How do you define AST transformations in Stratego?

- signatures
- rewrite rules
- rewrite strategies
- strategy combinators

Summary
lessons learned

How do you define AST transformations in Stratego?

- signatures
- rewrite rules
- rewrite strategies
- strategy combinators

What kind of strategies can you find in the Stratego library?

- arithmetics
- map, zip, foldr
- generic traversals

Literature

learn more

Literature

learn more

Spoofax

Lennart C. L. Kats, Eelco Visser: The Spoofax Language Workbench. Rules for Declarative Specification of Languages and IDEs. OOPSLA 2010

http://www.spoofax.org

Literature

learn more

Spoofax

Lennart C. L. Kats, Eelco Visser: The Spoofax Language Workbench. Rules for Declarative Specification of Languages and IDEs. OOPSLA 2010

http://www.spoofax.org

Stratego

Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, Eelco Visser: Stratego/XT 0.17. A language and toolset for program transformation. Science of Computer Programming, 72(1-2), 2008.

http://www.strategoxt.org

Outlook
coming next

declarative semantics definition

- Lecture 5: Static Analysis and Error Checking
- Lecture 6: Code Generation

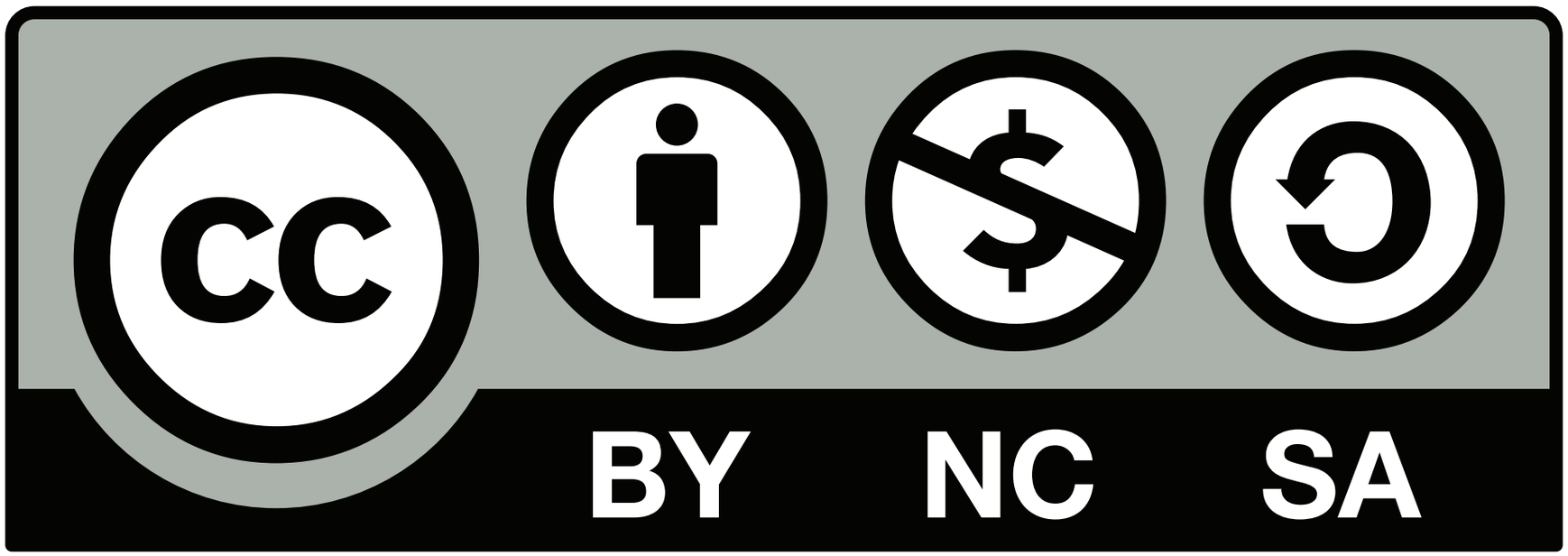
Assignment Oct 3: MiniJava grammar in SDF

- submit on Blackboard

Lab Oct 04

- pretty-printer for MiniJava
- syntactic editor services

copyrights


BY
NC
SA

Pictures

attribution & copyrights

Slides 1, 9, 10, 20, 27:
Stratego by Kendrick Arnett, some rights reserved

Slide 14:
Thesaurus by Enoch Lau, some rights reserved

Slide 21:
Google Maps Street View Camera by Ian Britton, some rights reserved

Slide 22:
Zipper by Rabensteiner, some rights reserved

Slide 23:
Fold by Kim Piper Werker, some rights reserved

Slide 24:
Inverse by lanchongzi, some rights reserved

Slide 25:
Maxwell's by Dominica Williamson, some rights reserved

Slide 58:
Reichstag by Wolfgang Staudt, some rights reserved