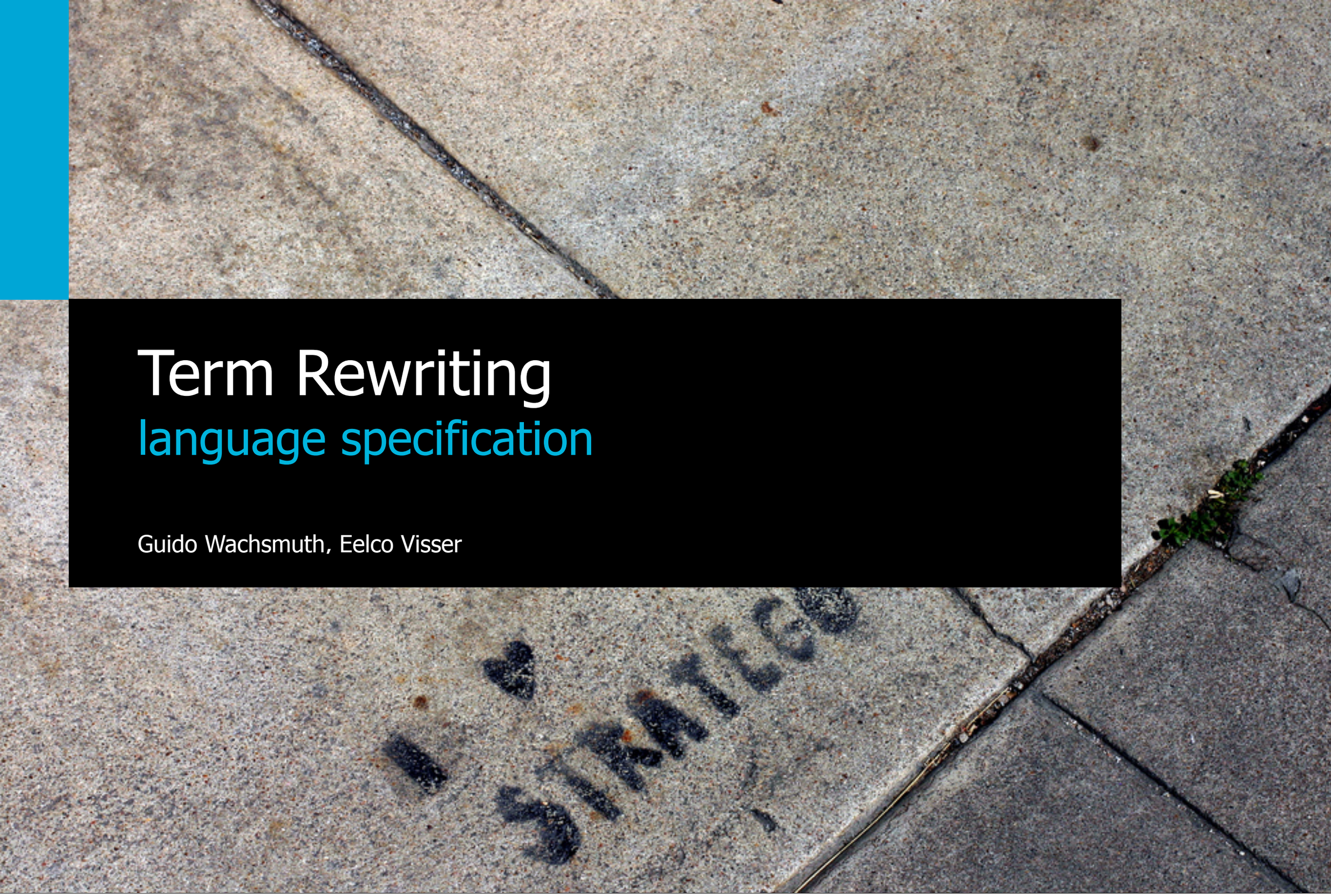
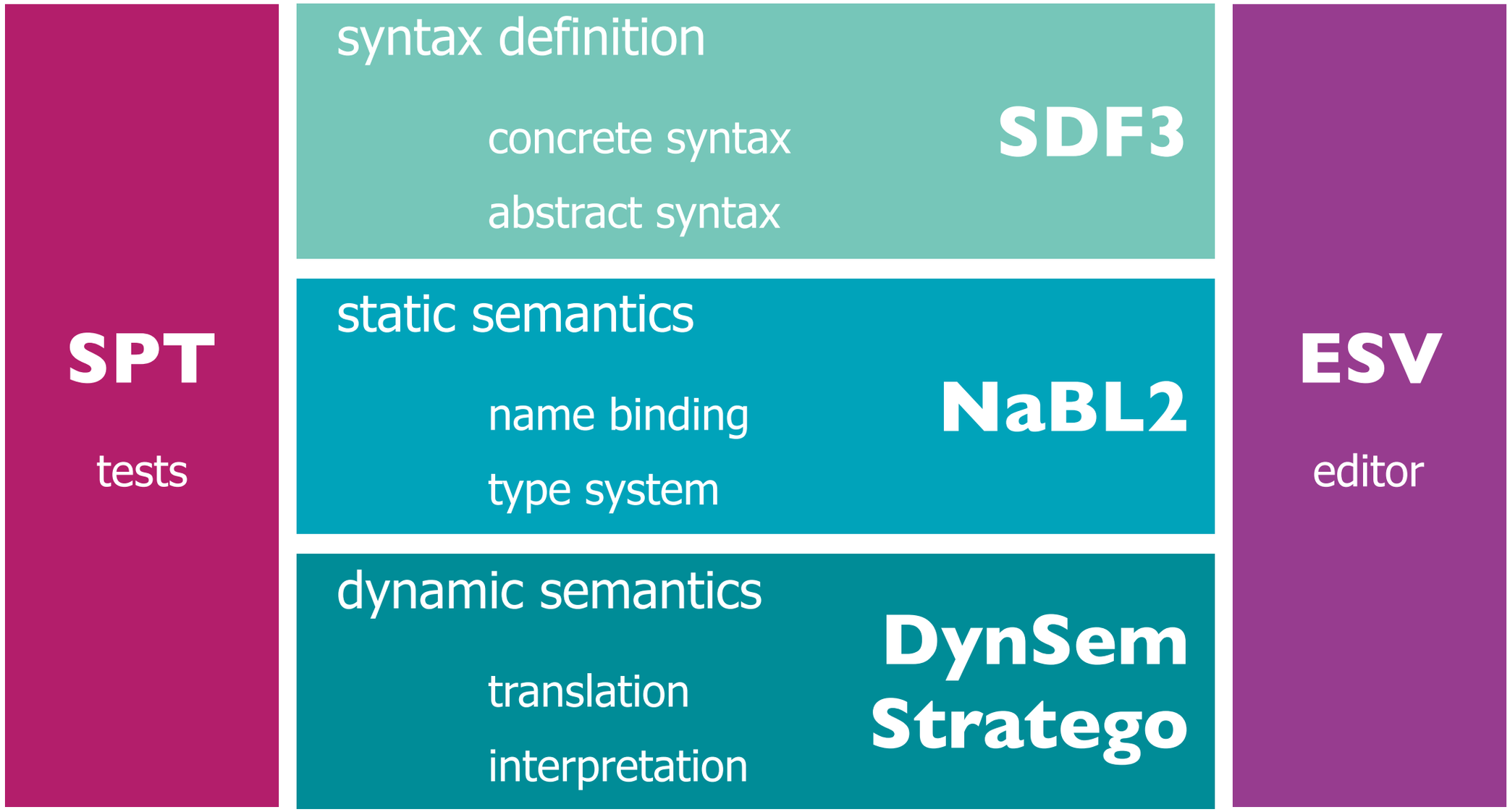
Term Rewriting
language specification

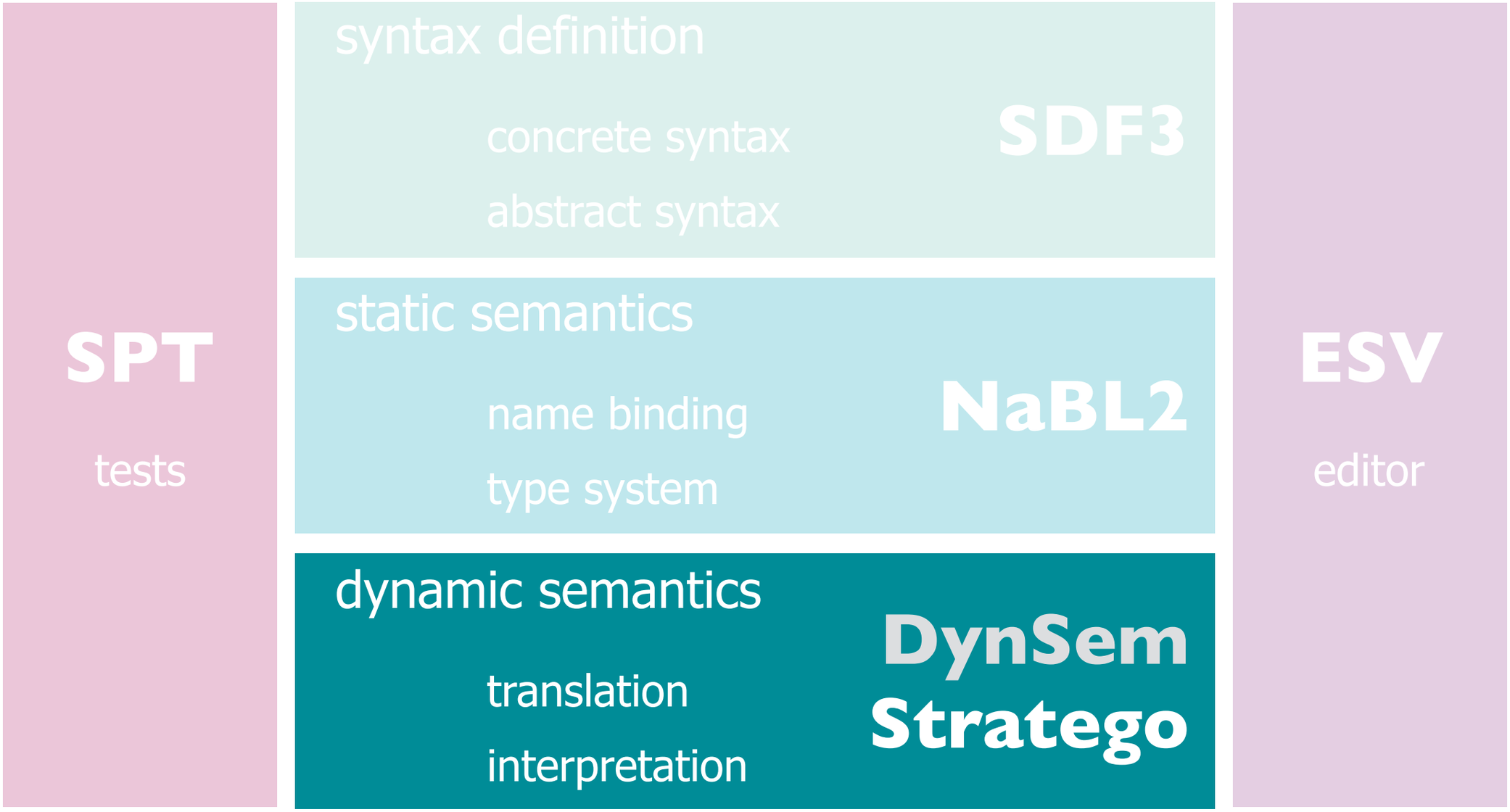
Guido Wachsmuth, Eelco Visser



TU Delft

IN4303 2015-2017
Compiler Construction


SPT
tests
syntax definition
concrete syntax
abstract syntax
SDF3
static semantics
name binding
type system
NaBL2
dynamic semantics
translation
interpretation
DynSem
Stratego
ESV
editor


SPT
tests
syntax definition
concrete syntax
abstract syntax
SDF3
static semantics
name binding
type system
NaBL2
dynamic semantics
translation
interpretation
DynSem
Stratego
ESV
editor

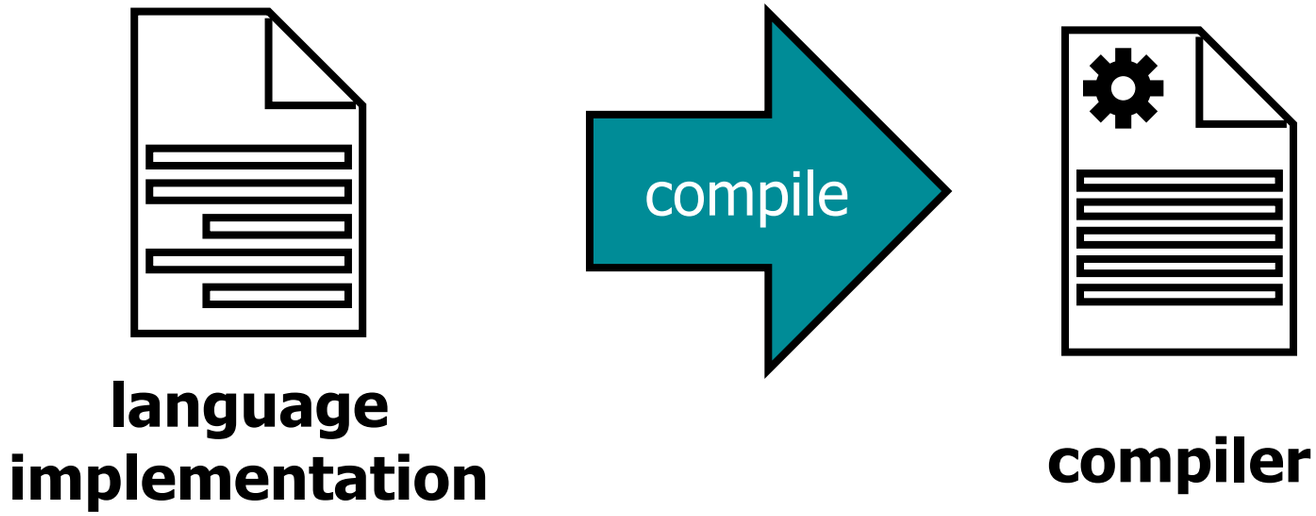
Language Software

compiler construction


meta language facility
THESAURUS

traditional compiler compilers

manual implementation



traditional compiler compilers

compilation + compilation

traditional compiler compilers

compilation + interpretation

Spoofax Language Workbench
manual implementation

Stratego

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

Stratego in Spoofax

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

Spoofax Language Workbench
manual implementation

Stratego

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

Stratego in Spoofax

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

Note: static analysis is replaced by the higher-level NaBL2 language

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

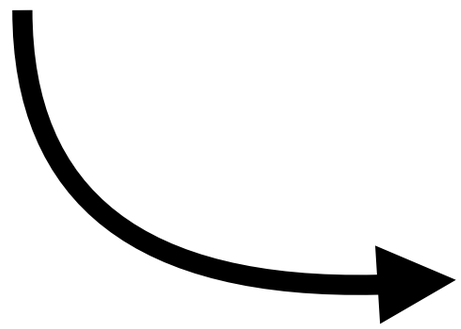
example

```
module desugar

imports

  include/Tiger
  operators

strategies

  desugar-all = innermost(desugar)

rules

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

Terms

Terms

```
let function fact(n : int) : int =
      if n < 1 then 1 else (n * fact(n - 1))
 in fact(10)
end
```



```
Let(
  [ FunDec(
      "fact"
    , [FArg("n", Tp(Tid("int")))]
    , Tp(Tid("int"))
    , If(
        Lt(Var("n"), Int("1"))
      , Int("1")
      , Seq(
          [ Times(
              Var("n")
            , Call(
                Var("fact")
              , [Minus(Var("n"), Int("1"))]
              )
            )
          ]
        )
      )
    )
  ]
, [Call(Var("fact"), [Int("10")])]
)
```

Syntax of Terms

```
module Terms

sorts Cons Term

lexical syntax

    Cons = [a-zA-Z][a-zA-Z0-9]*

context-free syntax

    Term.App = <<Cons>(<{Term ","}*>)>
```

```
Zero()

Succ(Zero())

Cons(A(), Cons(B(), Nil()))
```

Syntax of Terms

```
module Terms

sorts Cons Term

lexical syntax

    Cons = [a-zA-Z][a-zA-Z0-9]*

context-free syntax

    Term.App   = <<Cons>(<{Term ","}*>)>
    Term.List  = <[<{Term ","}*>]>
    Term.Tuple = <(<{Term ","}*>)>
```

```
Zero()

Succ(Zero())

[A(), B()]
```

Syntax of Terms

```
module ATerms

sorts Cons Term

lexical syntax
   Cons       = [a-zA-Z][a-zA-Z0-9]*
   Cons       = String
   Int        = [0-9]+
   String     = "\"" StringChar* "\""
   StringChar = ~[\"\\n]
   StringChar = "\\" [\"\\]

context-free syntax
   Term.Str   = <<String>>
   Term.Int   = <<Int>>
   Term.App   = <<Cons>(<{Term ","}*>)>
   Term.List  = <[<{Term ","}*>]>
   Term.Tuple = <(<{Term ","}*>)>
```

```
0

1

[A(), B()]

Var(“x\\”)

Let(
  [ Decl(“x”, IntT()),
    Decl(“y”, BoolT())
  ]
  , Eq(Var(“x”), Int(0))
)
```

Syntax of A(nnotated)Terms

```
module ATerms

sorts Cons Term

lexical syntax
   Cons = [a-zA-Z][a-zA-Z0-9]*
   // more lexical syntax omitted

context-free syntax

   Term.Anno     = <<PreTerm>{<{Term “,"}*>}>
   Term          = <<PreTerm>>

   PreTerm.Str   = <<String>>
   PreTerm.Int   = <<Int>>
   PreTerm.App   = <<Cons>(<{Term ","}*>)>
   PreTerm.List  = <[<{Term ","}*>]>
   PreTerm.Tuple = <(<{Term ","}*>)>
```

```
Var(“x”){Type(IntT())}
```

Signatures

Signatures

```
context-free syntax

  Exp.Uminus  = [- [Exp]]
  Exp.Power   = [[Exp] ** [Exp]]
  Exp.Times   = [[Exp] *  [Exp]]
  Exp.Divide  = [[Exp] /  [Exp]]
  Exp.Plus    = [[Exp] +  [Exp]]
  Exp.Minus   = [[Exp] -  [Exp]]
  Exp.CPlus   = [[Exp] +i [Exp]]
  Exp.CMinus  = [[Exp] -i [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.True    = <true>
  Exp.False   = <false>
  Exp.And     = [[Exp] & [Exp]]
  Exp.Or      = [[Exp] | [Exp]]
```

```
signature
  constructors
    Uminus    : Exp -> Exp
    Power     : Exp * Exp -> Exp
    Times     : Exp * Exp -> Exp
    Divide    : Exp * Exp -> Exp
    Plus      : Exp * Exp -> Exp
    Minus     : Exp * Exp -> Exp
    CPlus     : Exp * Exp -> Exp
    CMinus    : 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
    True      : Exp
    False     : Exp
    And       : Exp * Exp -> Exp
    Or        : Exp * Exp -> Exp
```

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")]
   )
)
```

e1

e2

pattern matching

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

pattern instantiation

```
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

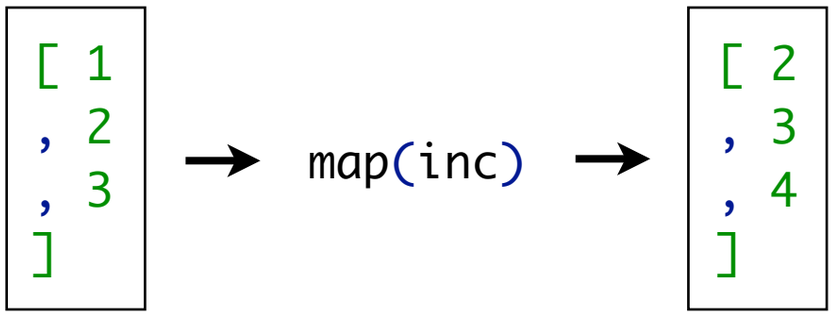
```
f(x,y): lhs -> rhs
```

- optional term parameters

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

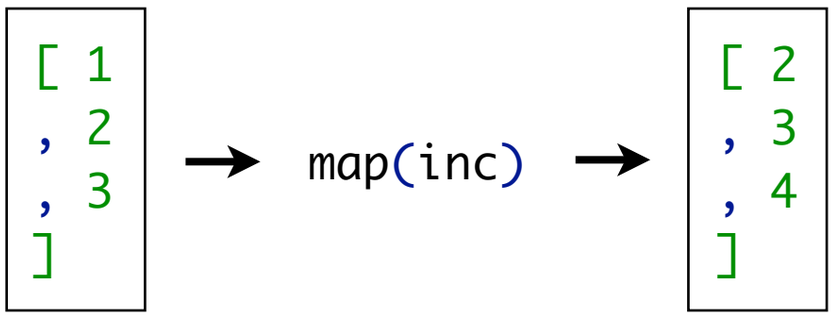
Rules with Parameters

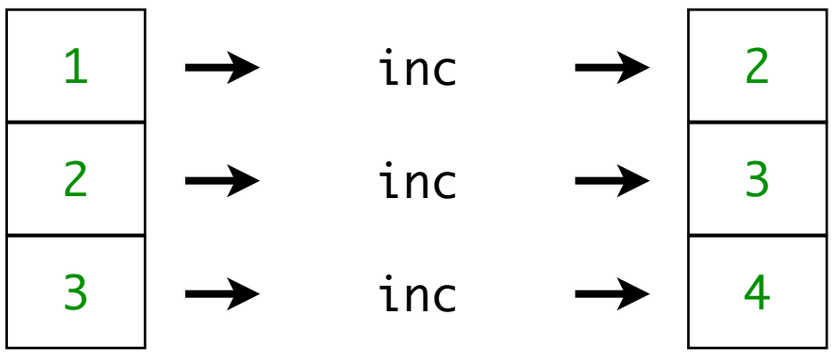
example: map



Rules with Parameters

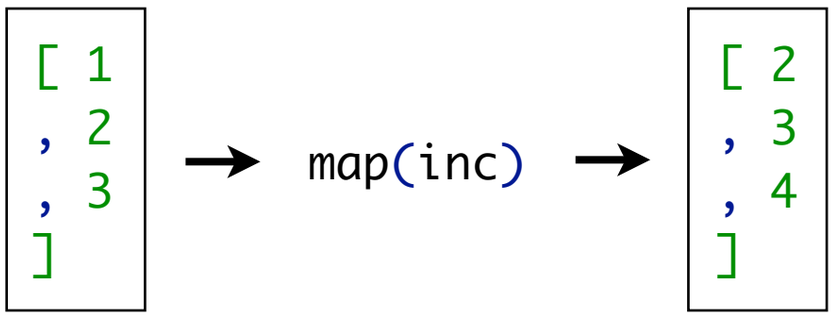
example: map

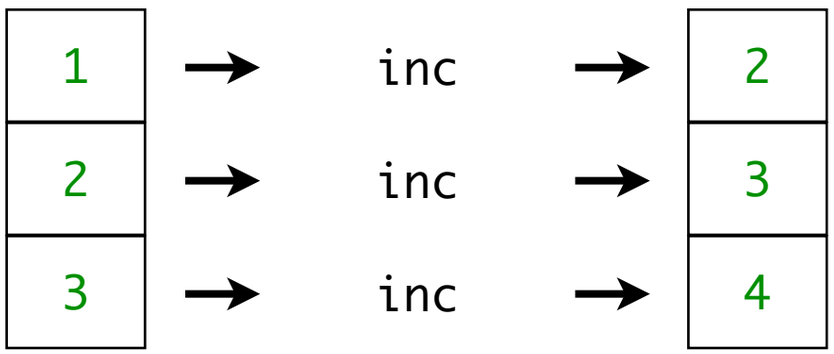




Rules with Parameters

example: map

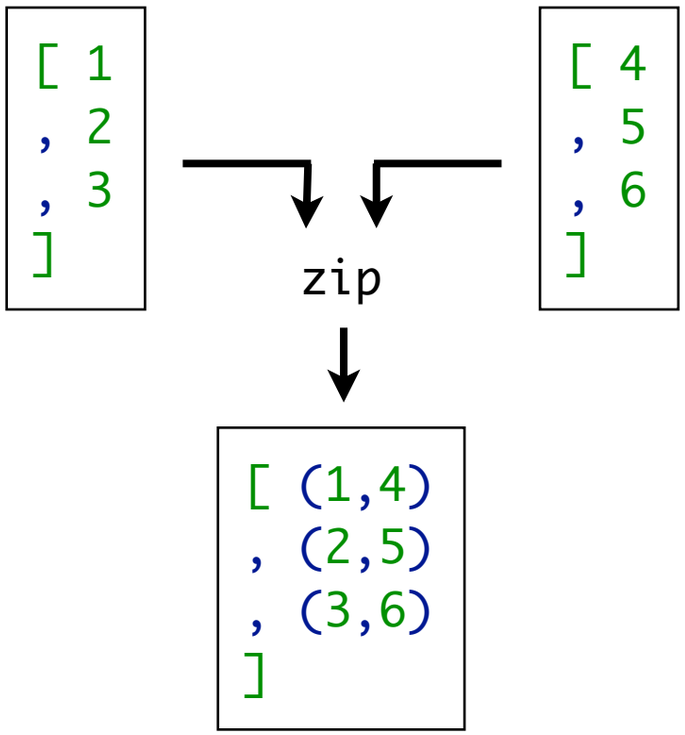





```
map(s): []     -> []
map(s): [x|xs] -> [<s> x | <map(s)> xs]
```

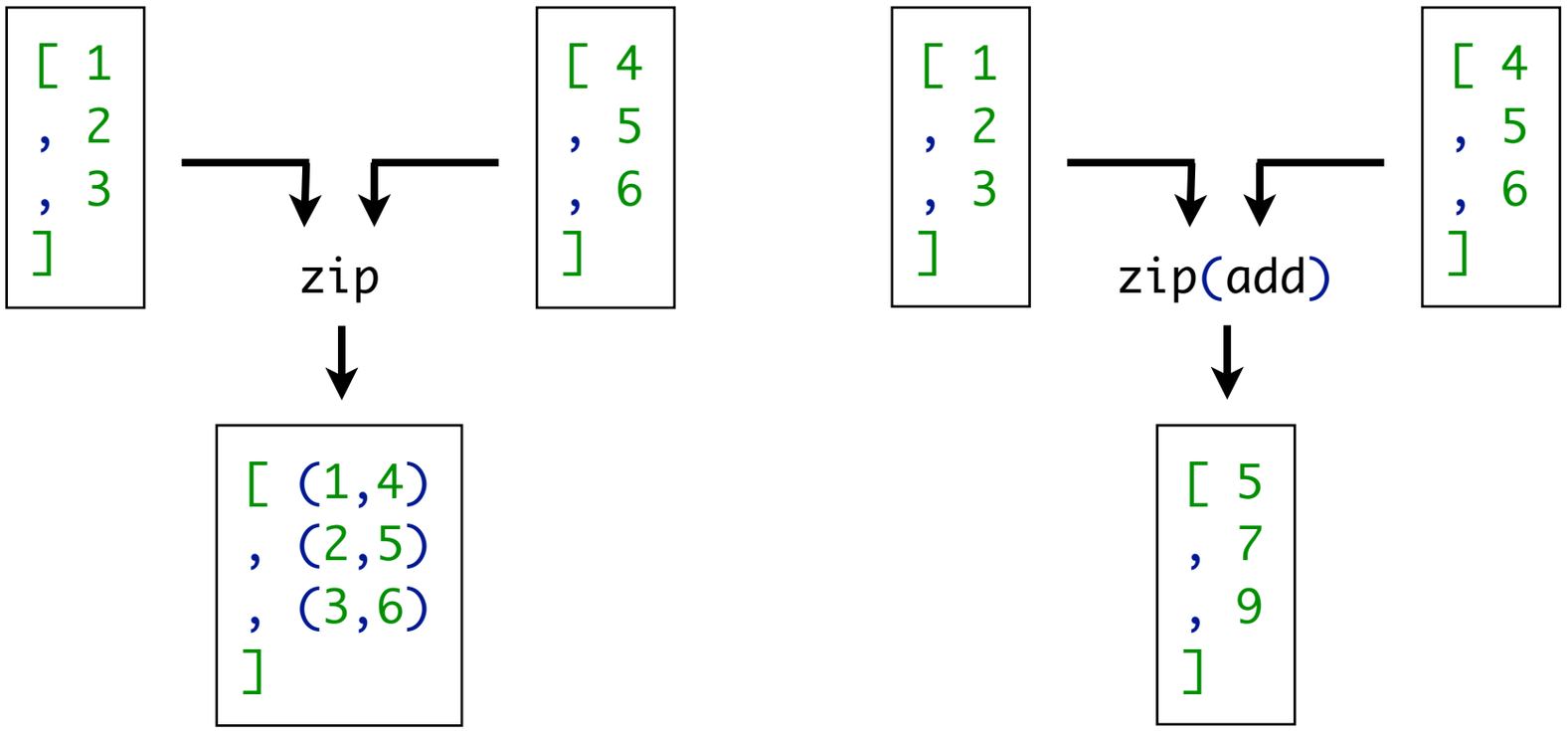
Rules with Parameters

example: zip



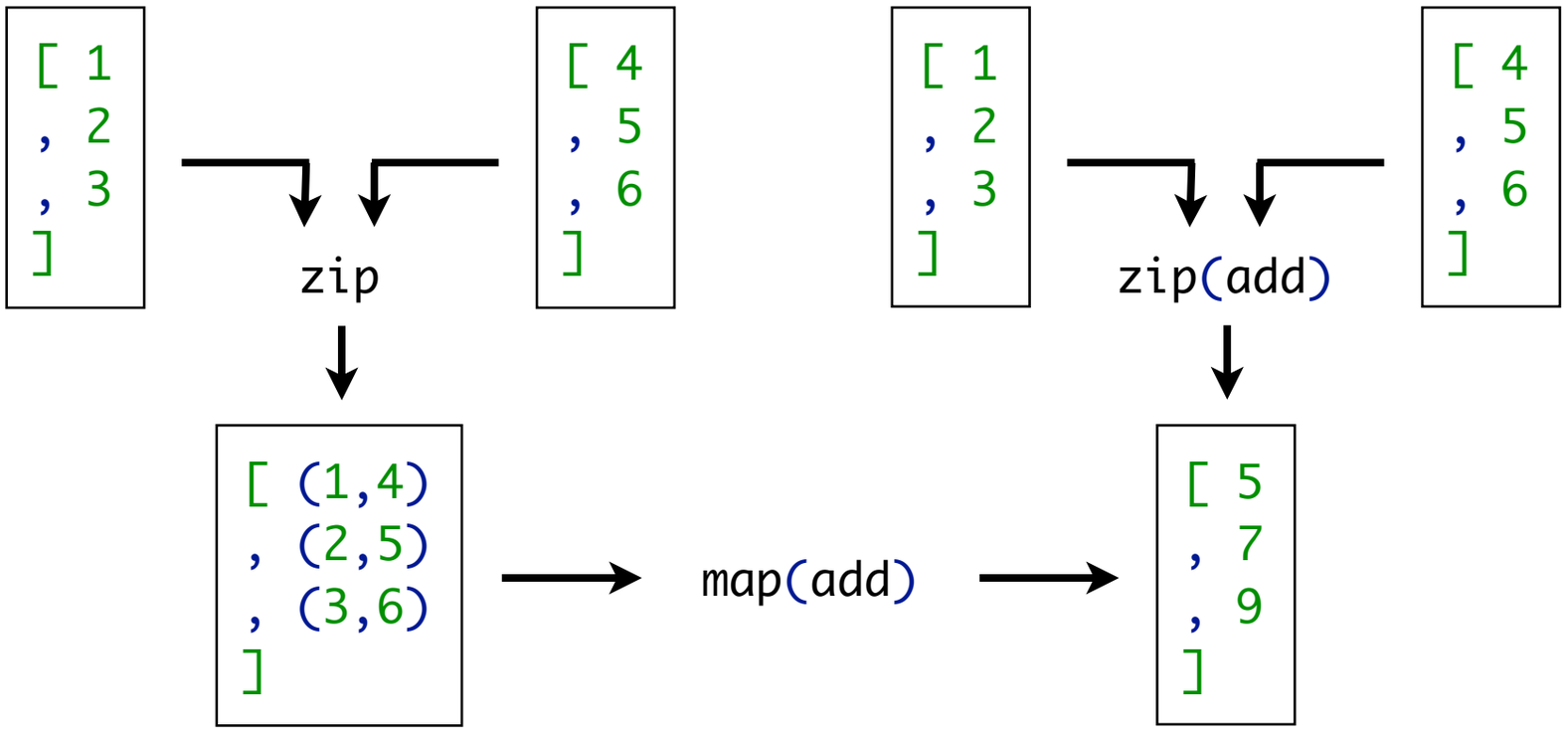
Rules with Parameters

example: zip



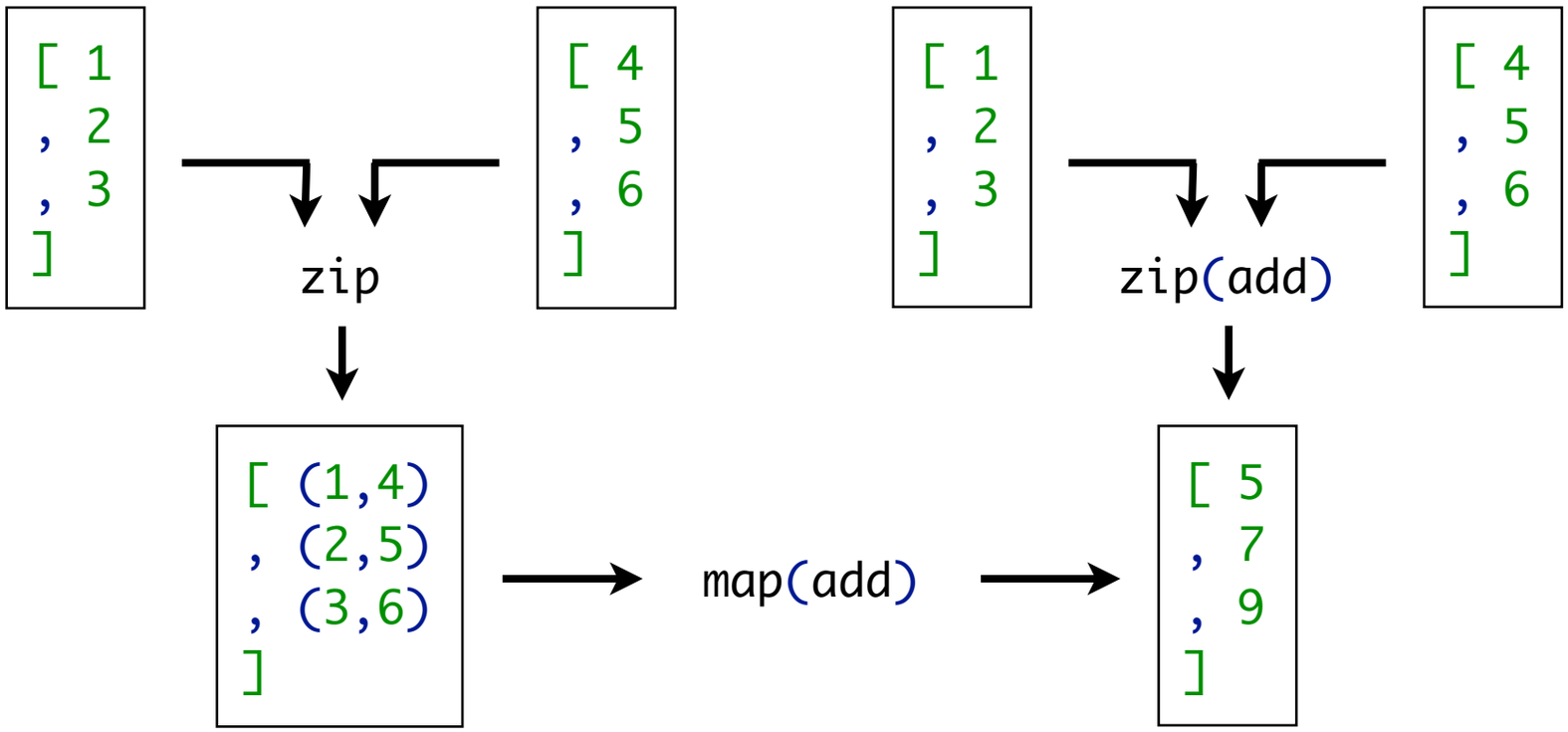
Rules with Parameters

example: zip



Rules with Parameters

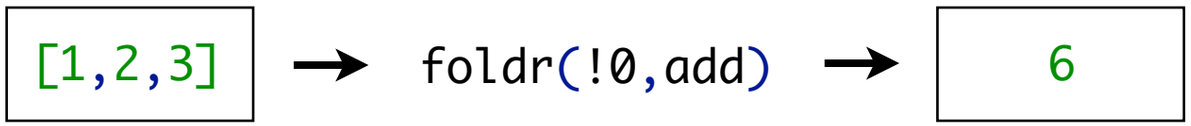
example: zip

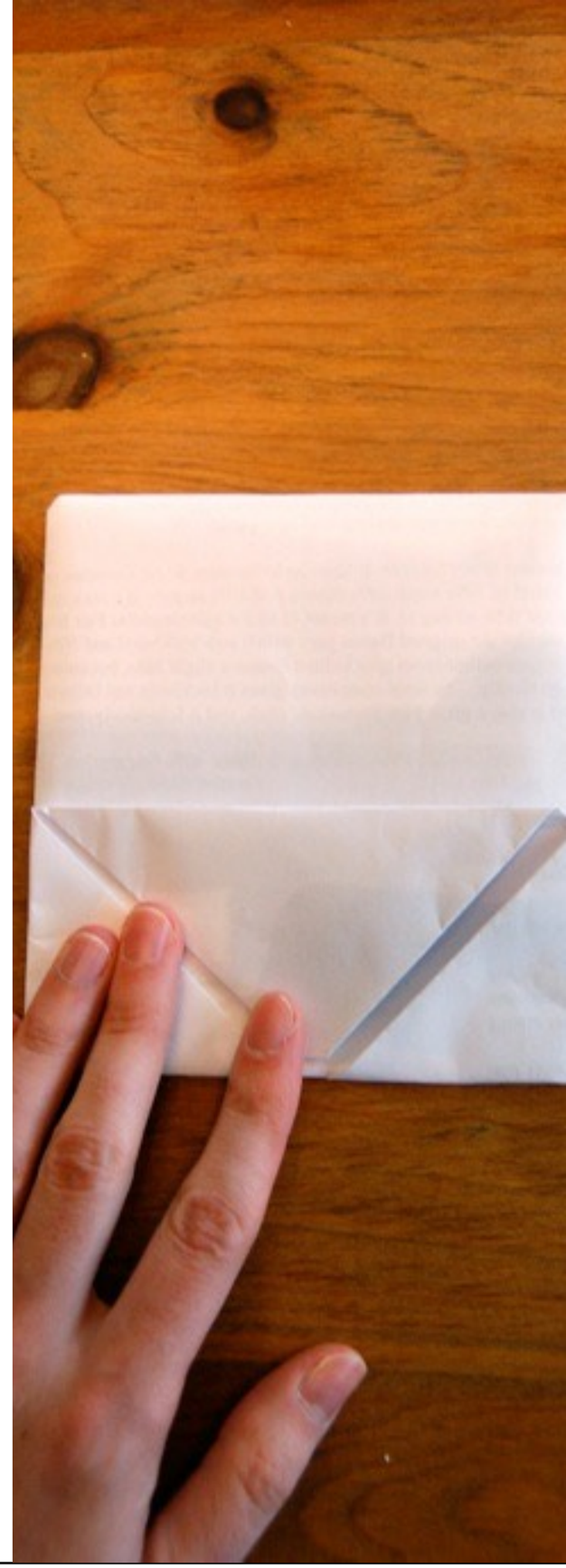



```
zip(s): ([],[])          -> []
zip(s): ([x|xs],[y|ys]) -> [<s> (x,y) | <zip(s)> (xs,ys)]

zip = zip(id)
```

Rules with Parameters

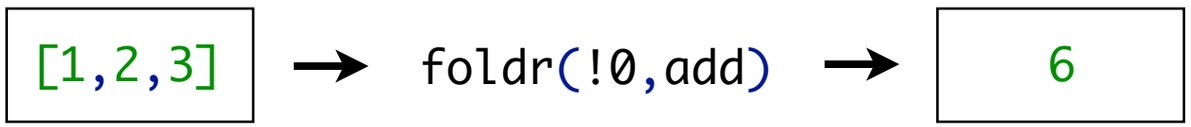
example: fold

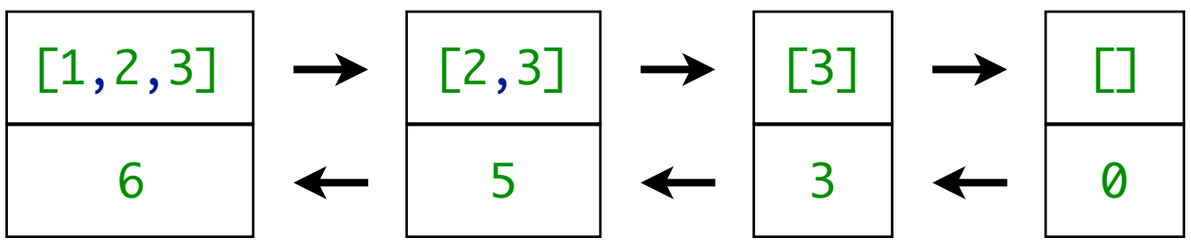


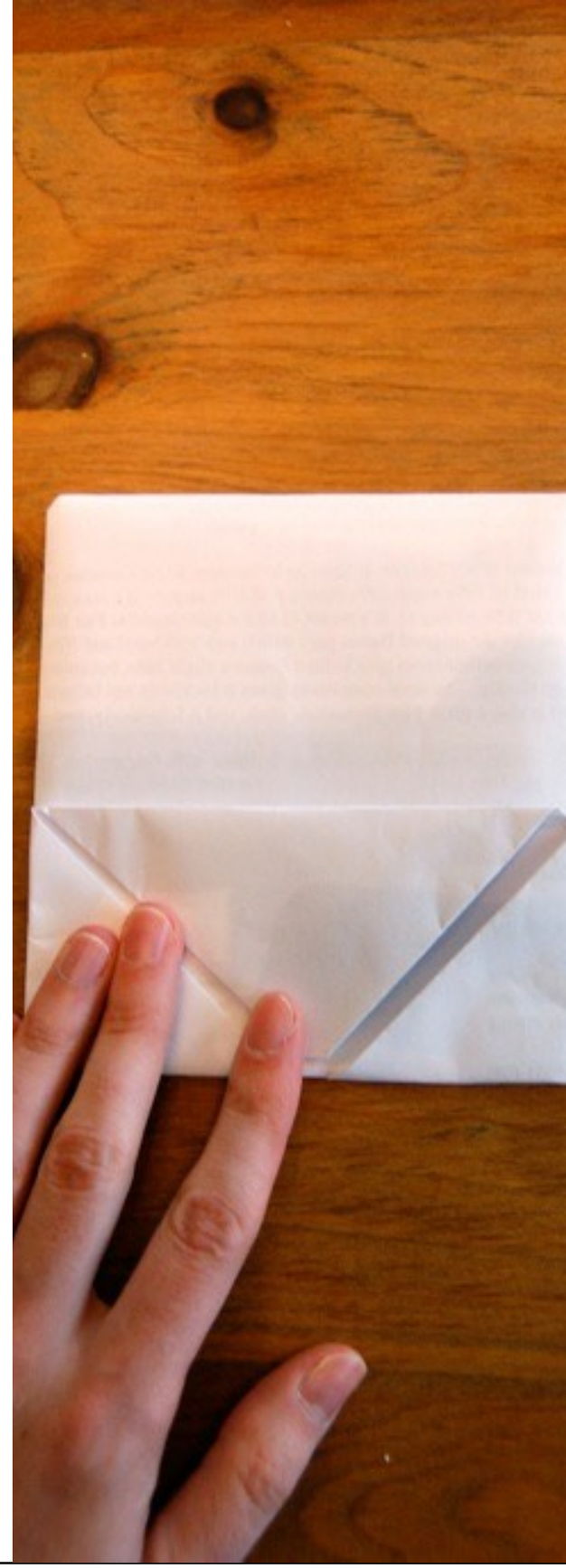


Rules with Parameters

example: fold

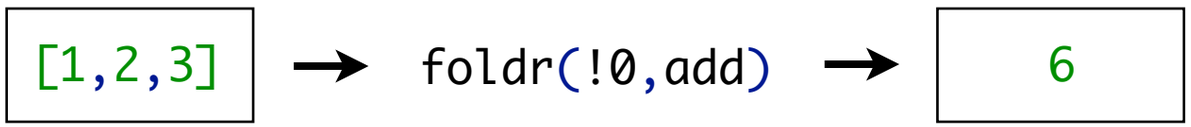


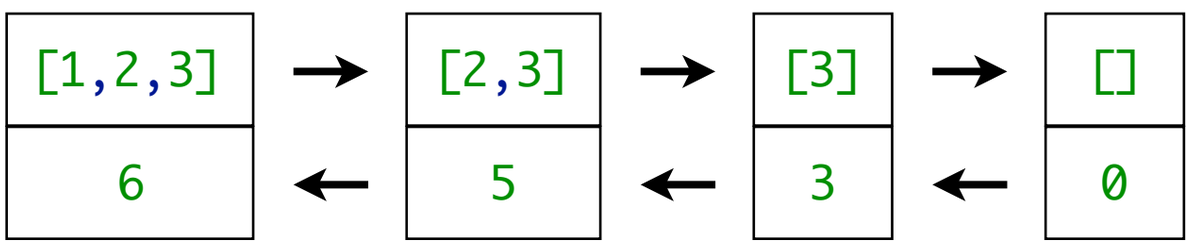




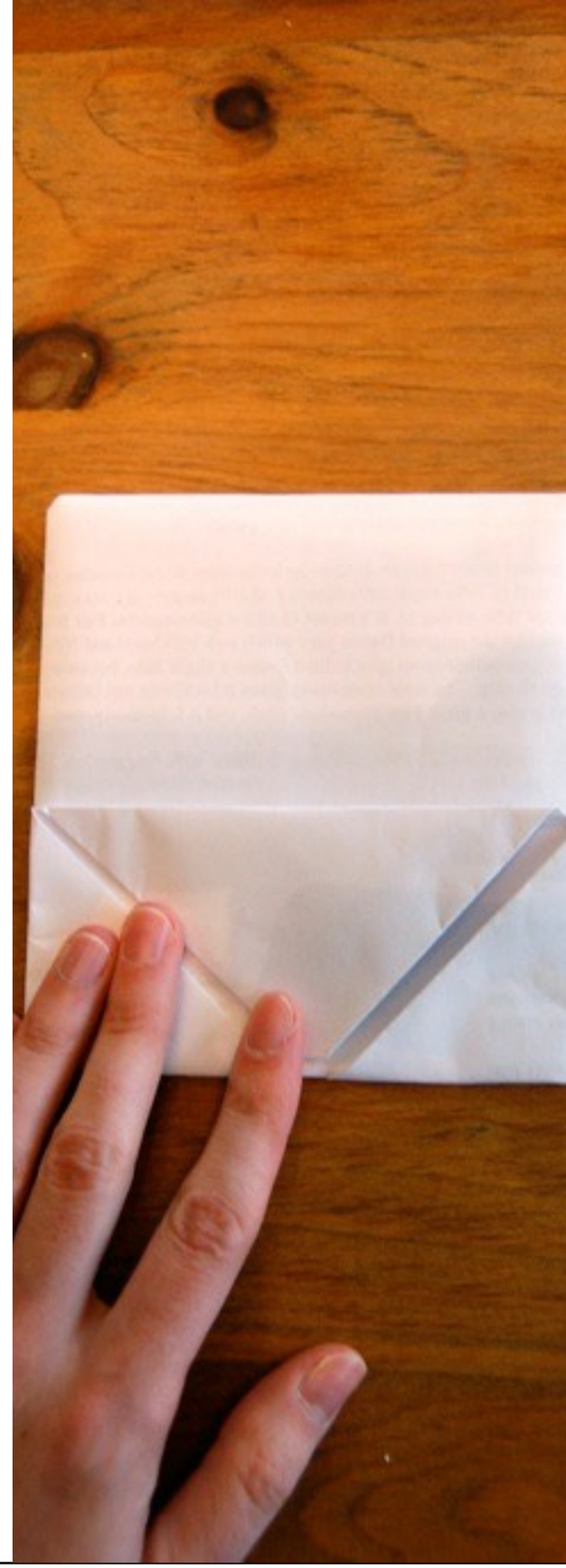
Rules with Parameters

example: fold






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

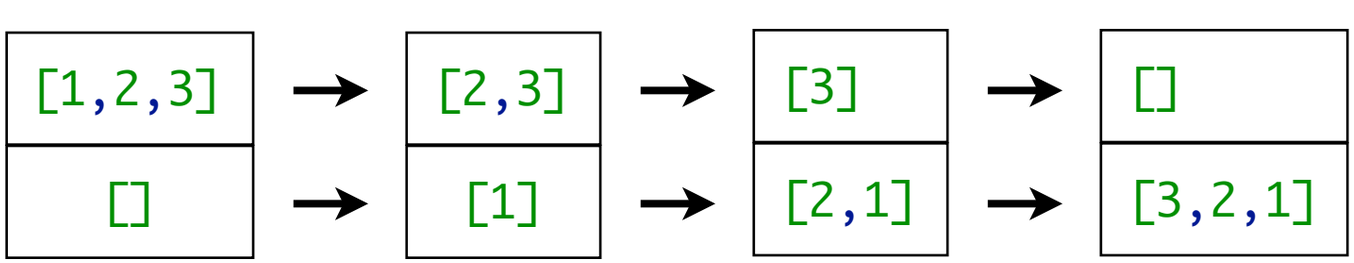


Rules with Parameters

example: inverse

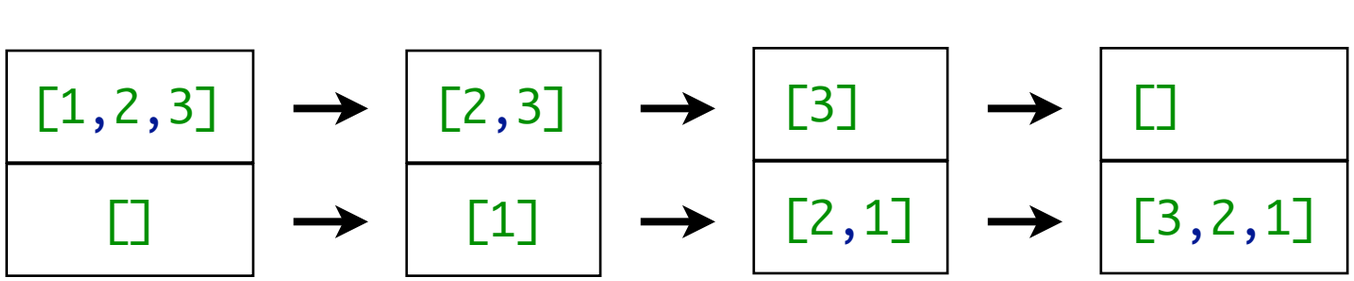
Rules with Parameters

example: inverse



Rules with Parameters

example: inverse




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

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
```

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
```

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)))
```

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))
```

Strategy Combinators

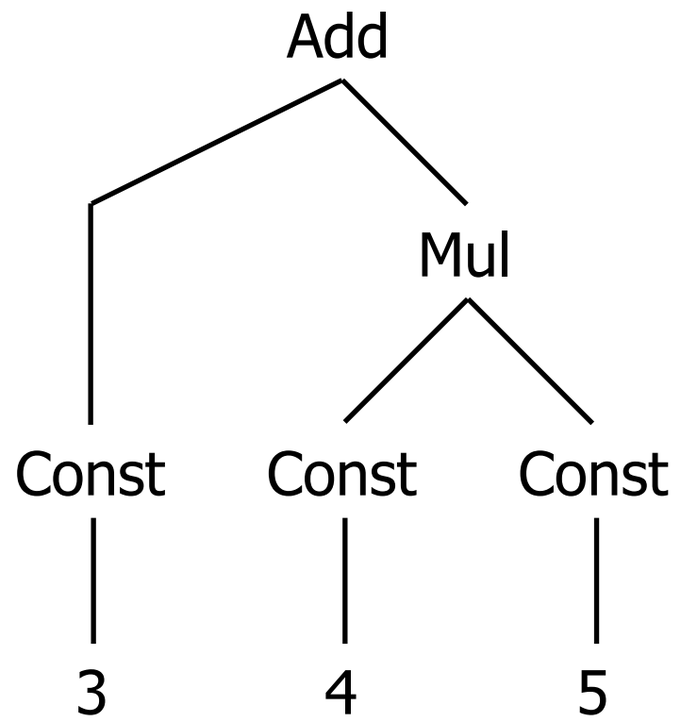
examples

```
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

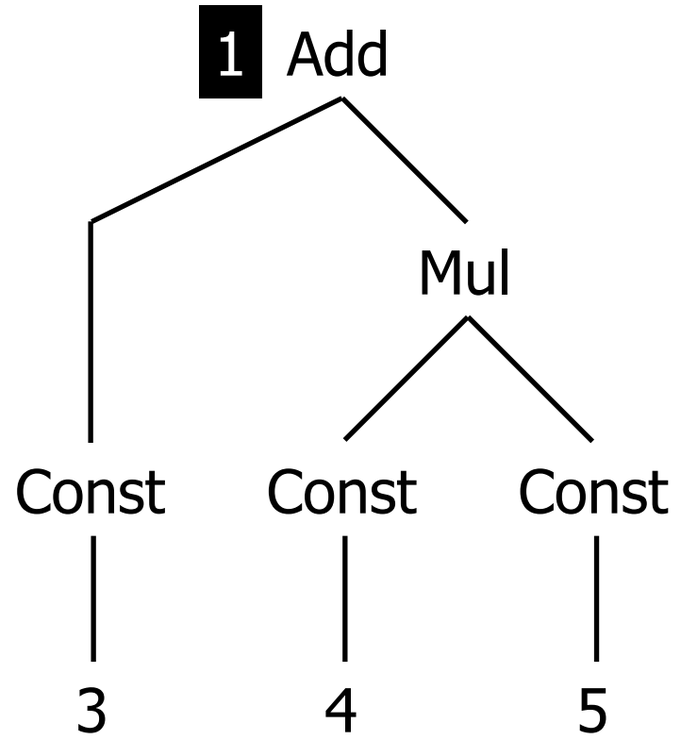
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

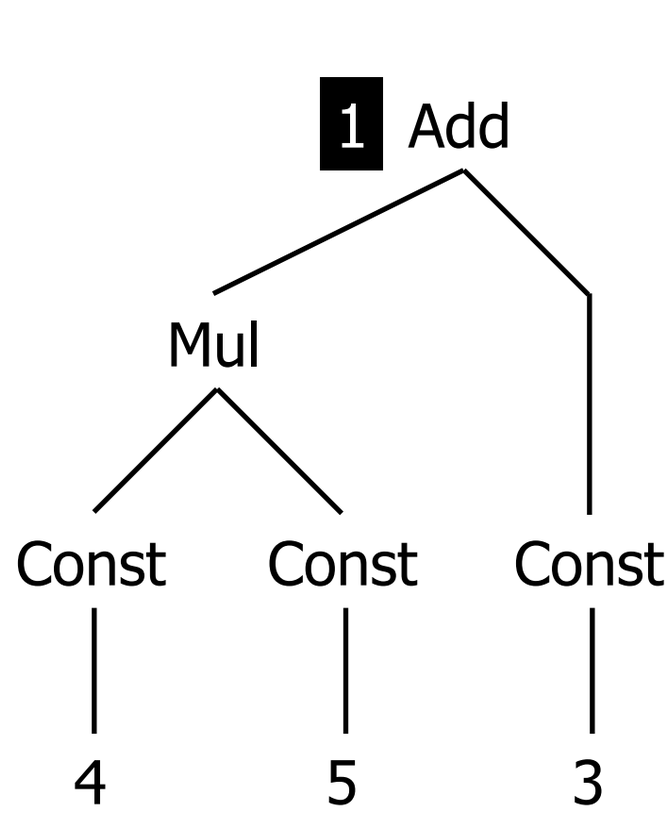
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

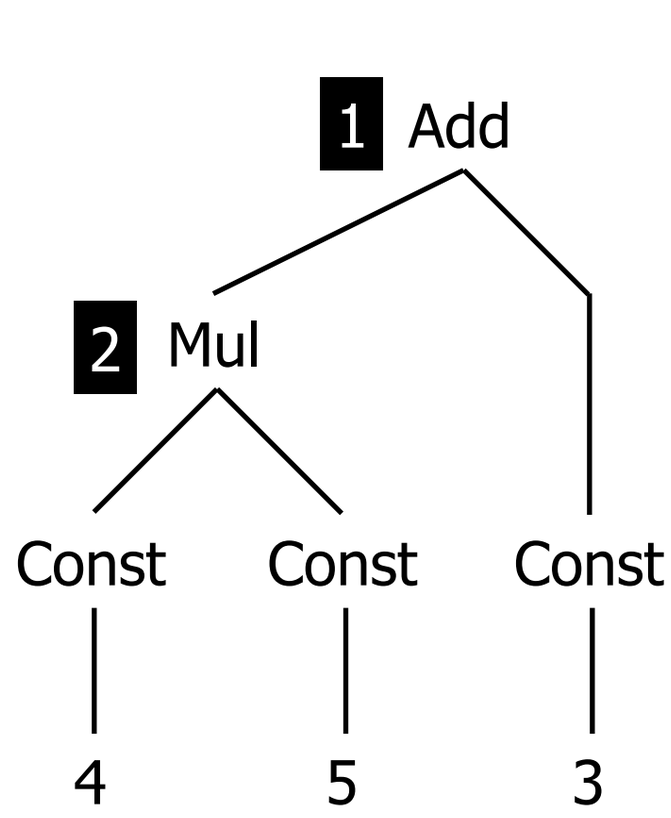
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

topdown(switch)
```



Stratego

example

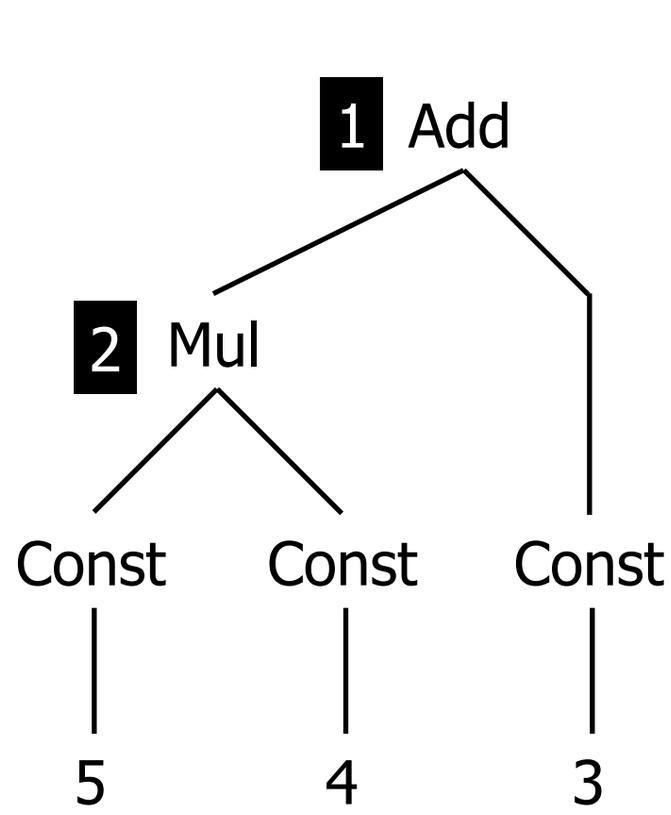
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

topdown(switch)
```



Stratego

example

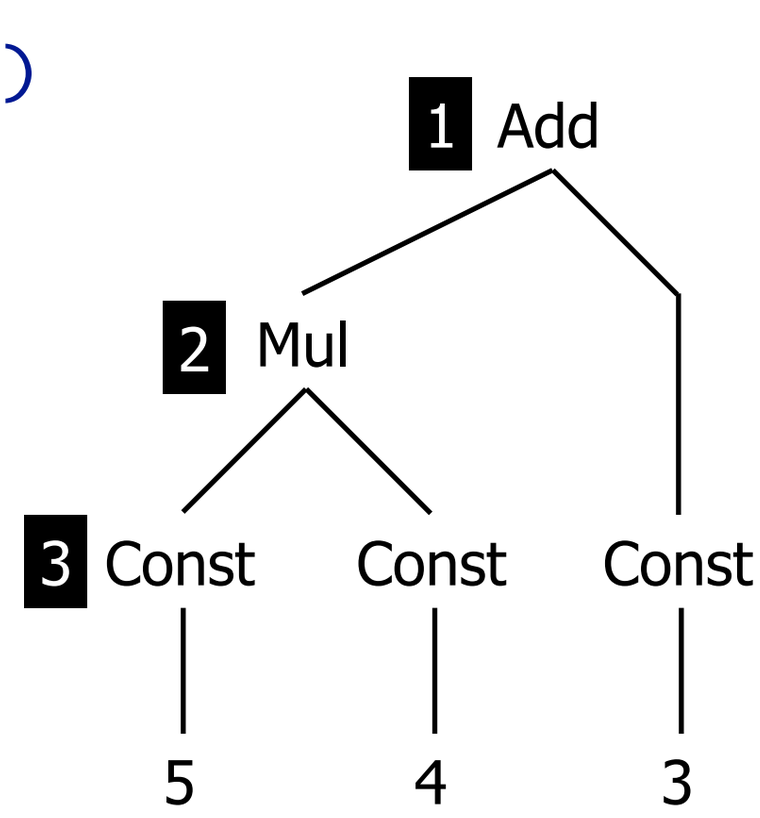
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

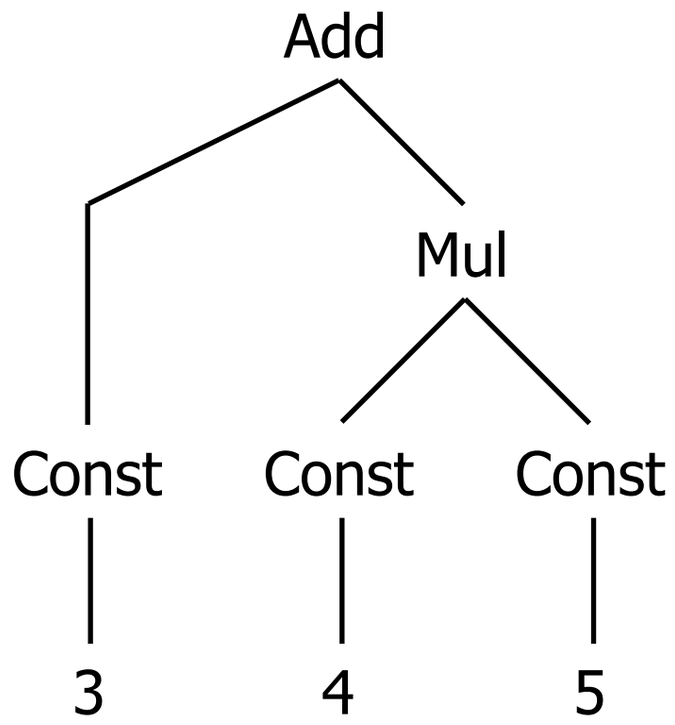
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

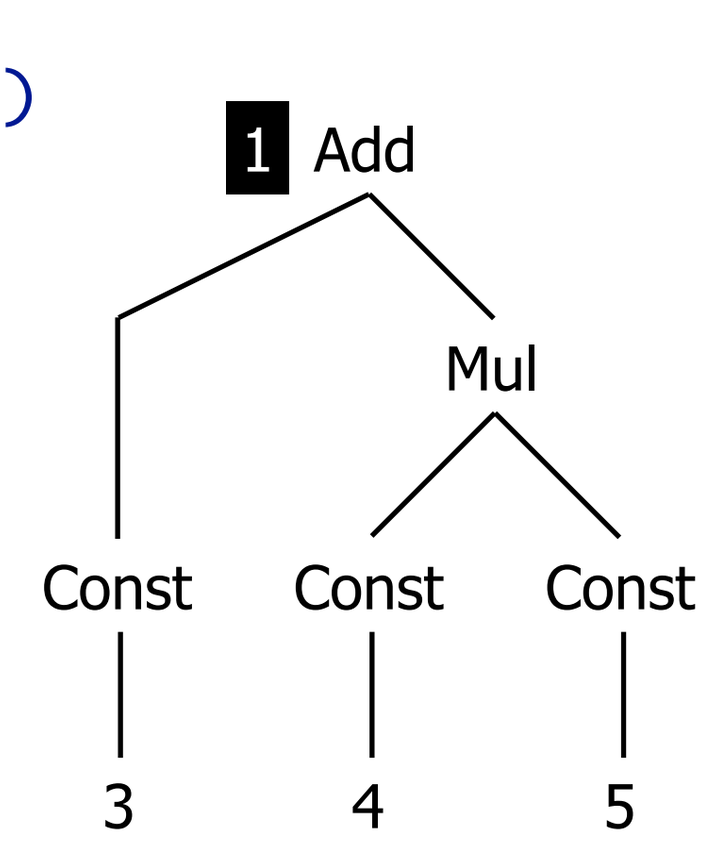
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

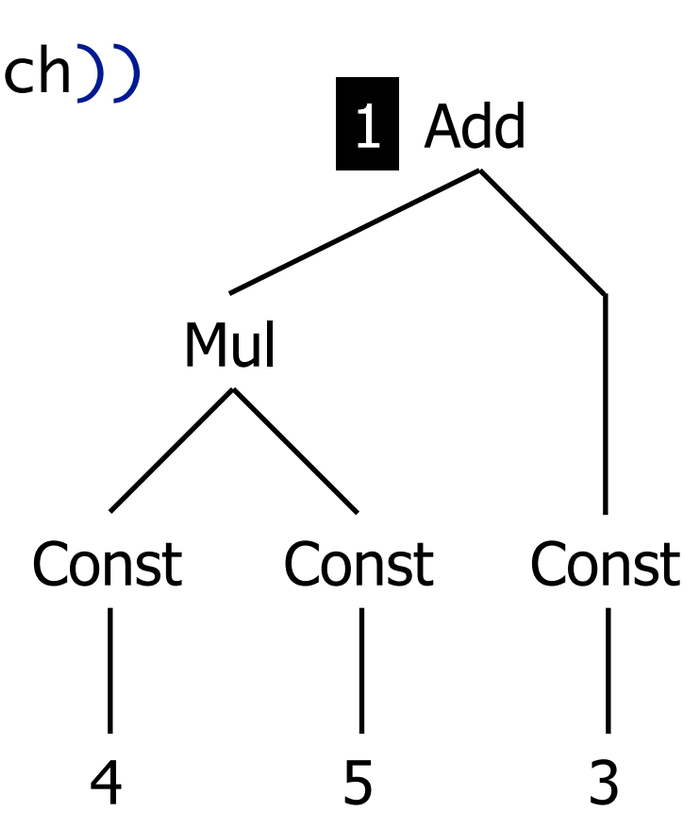
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

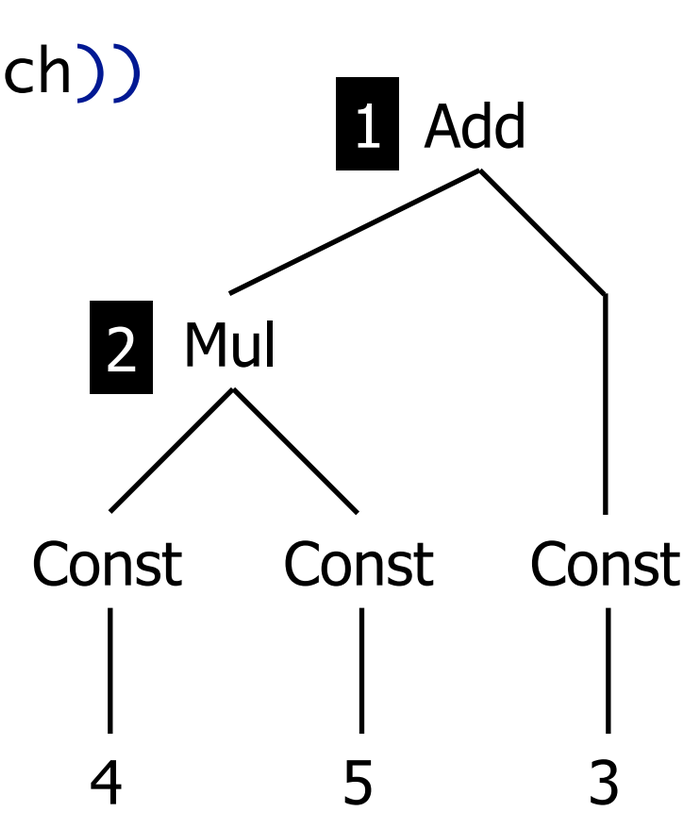
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

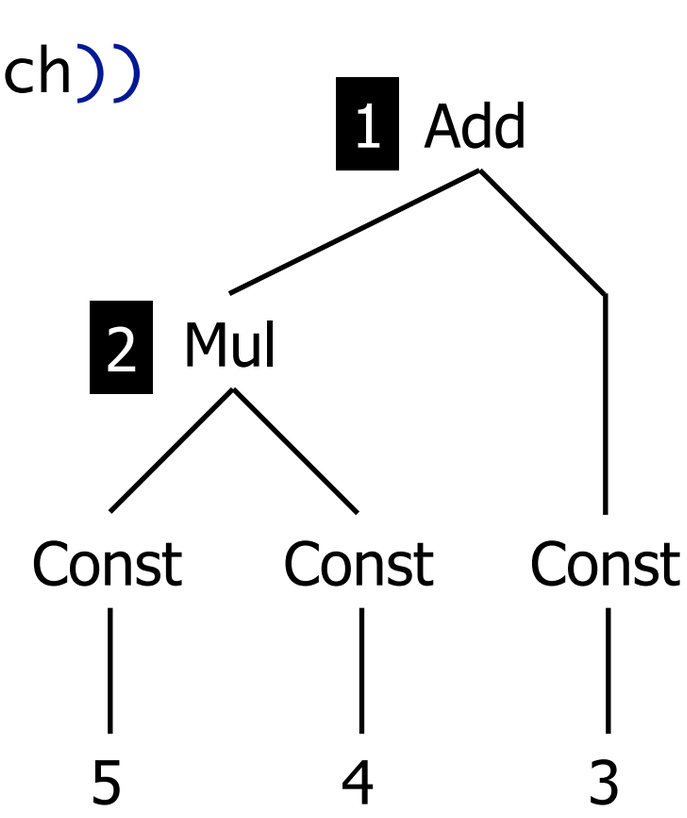
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

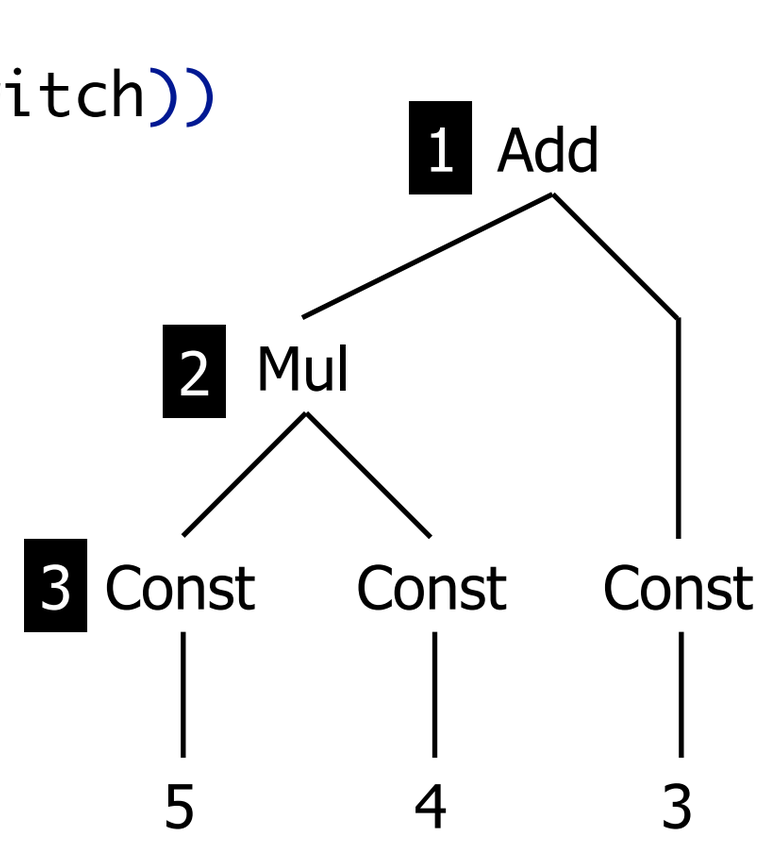
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

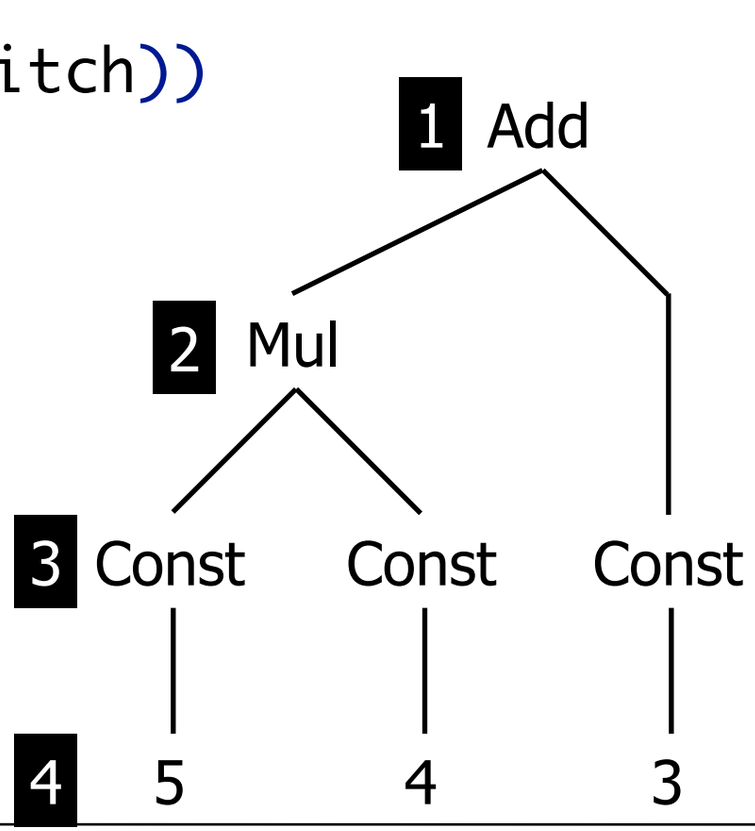
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

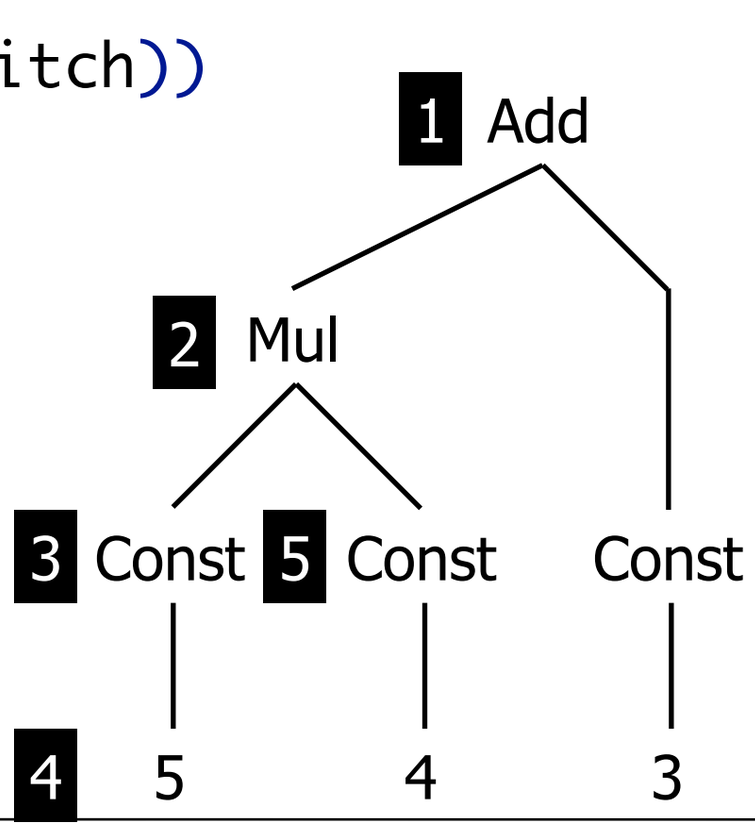
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

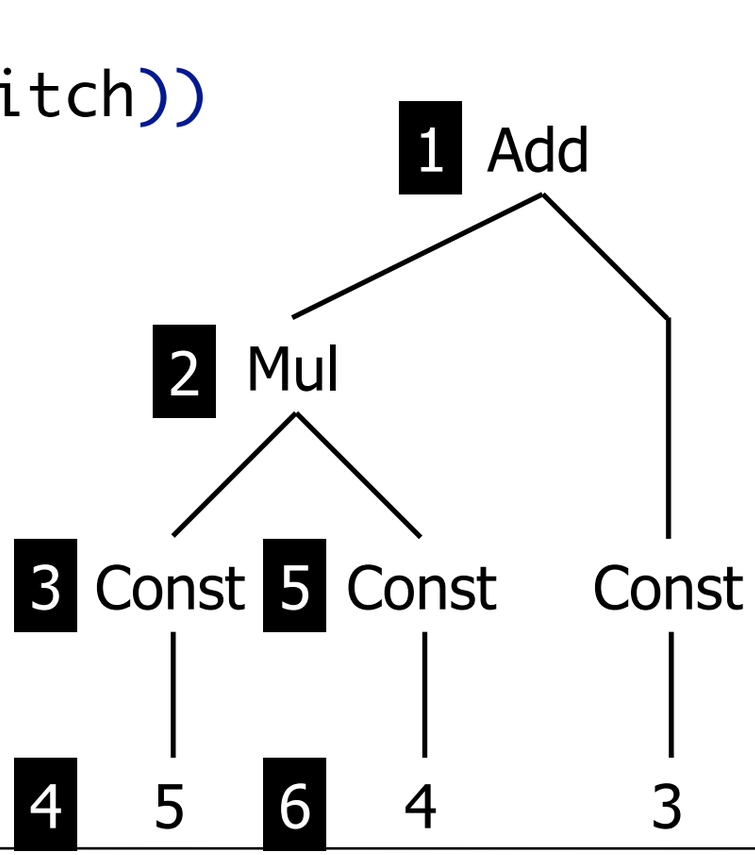
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

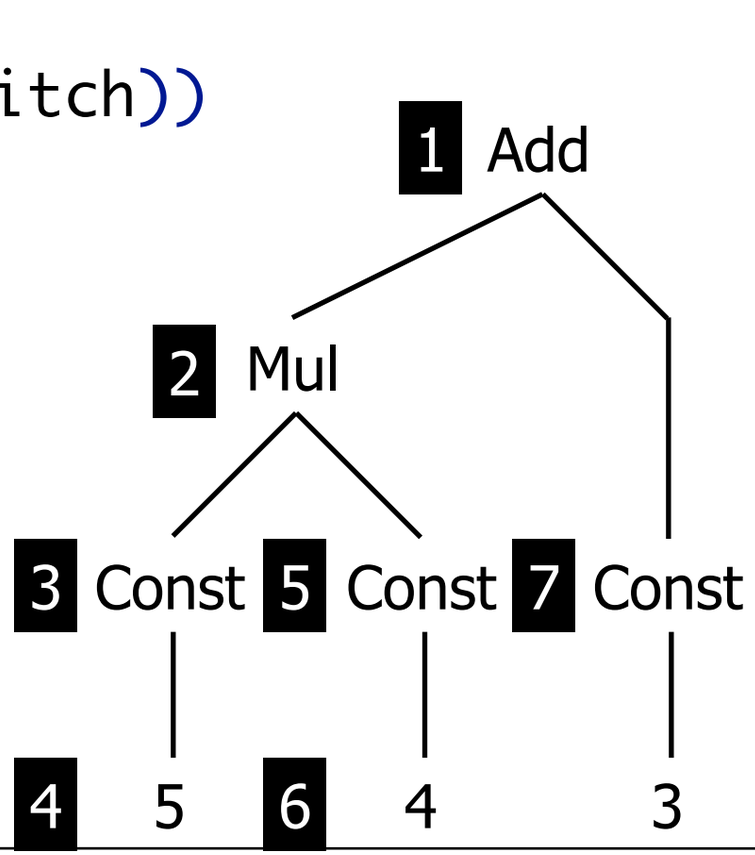
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

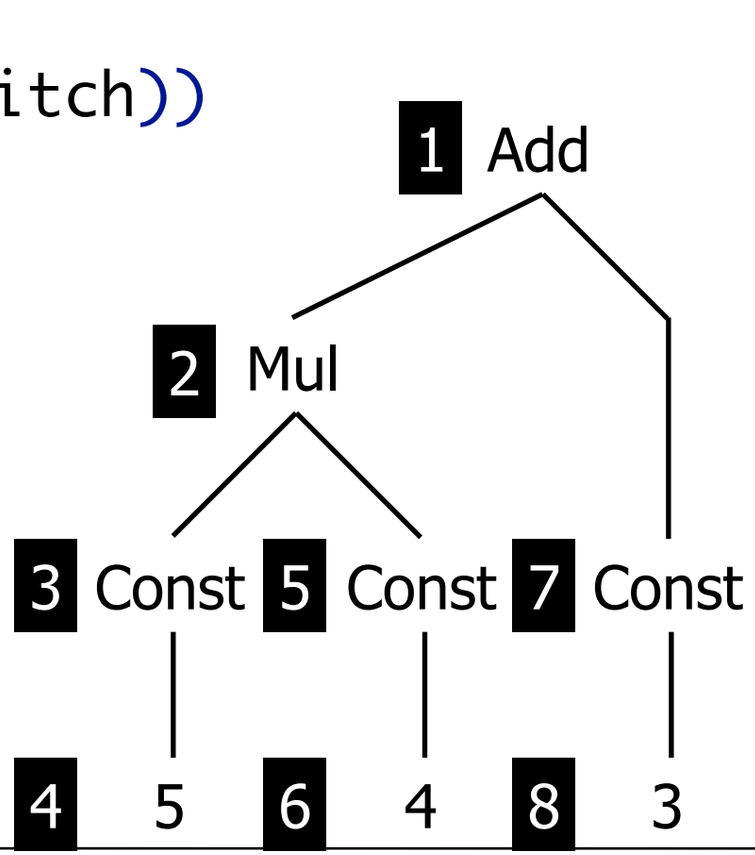
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

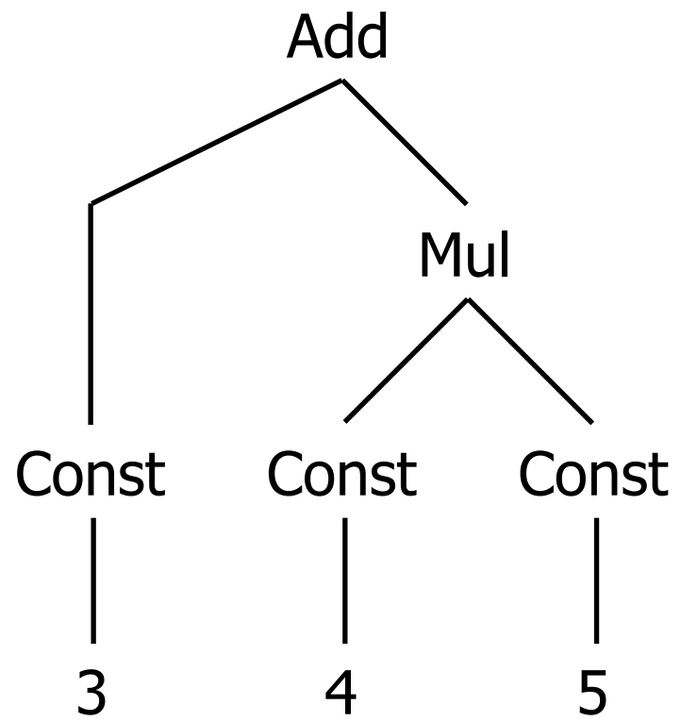
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

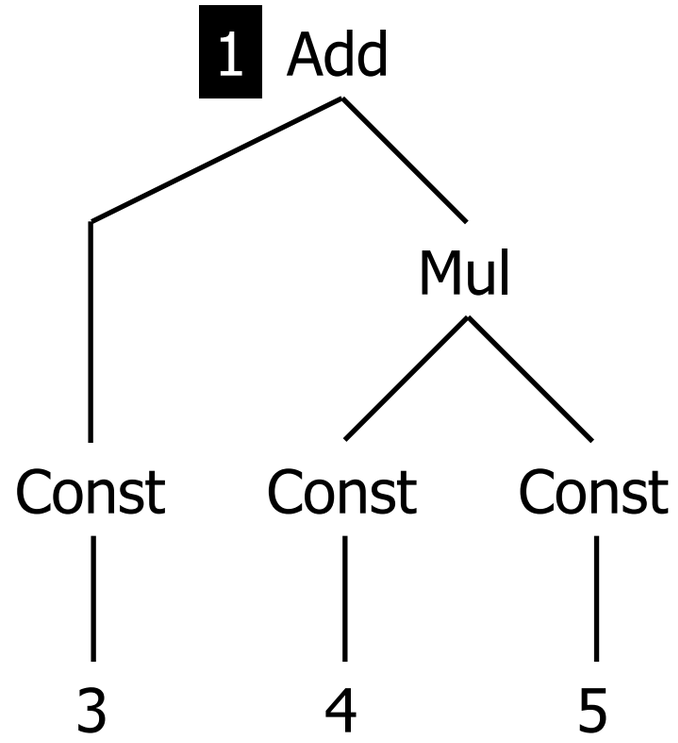
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

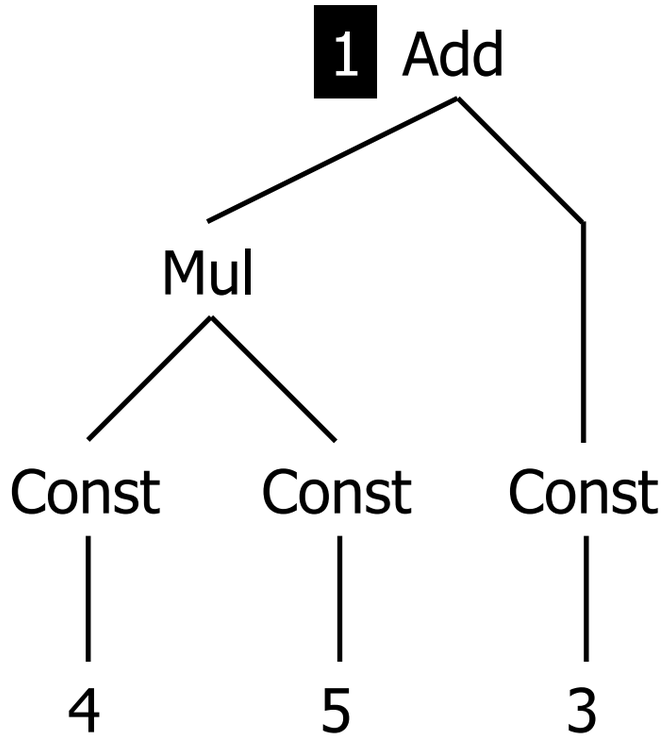
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

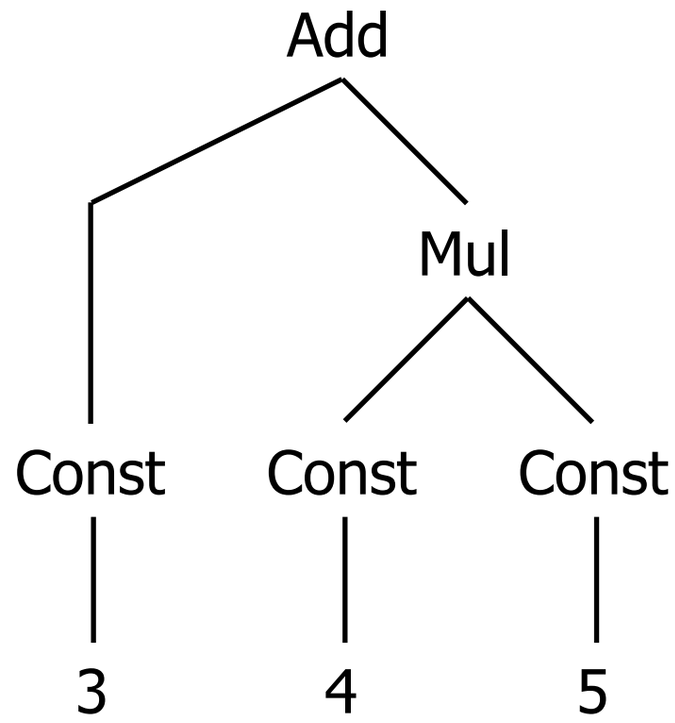
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

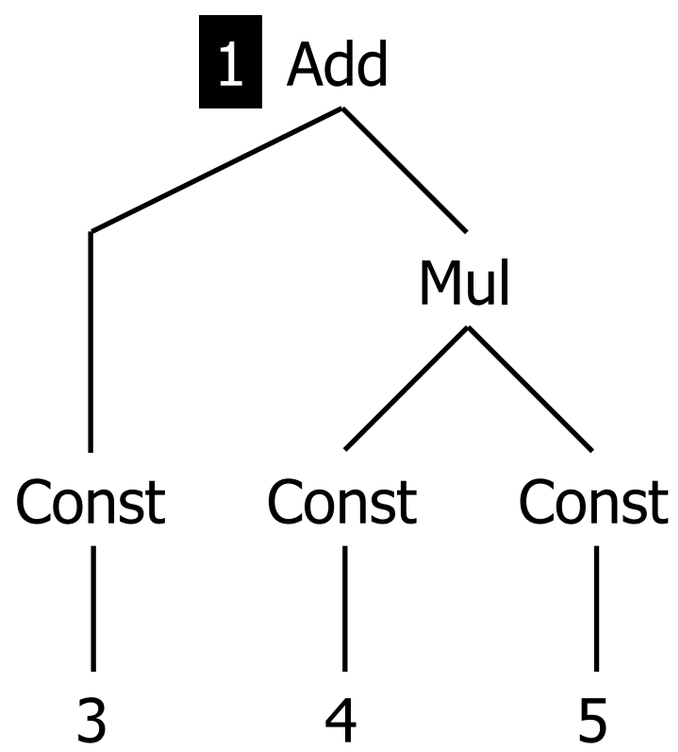
```
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

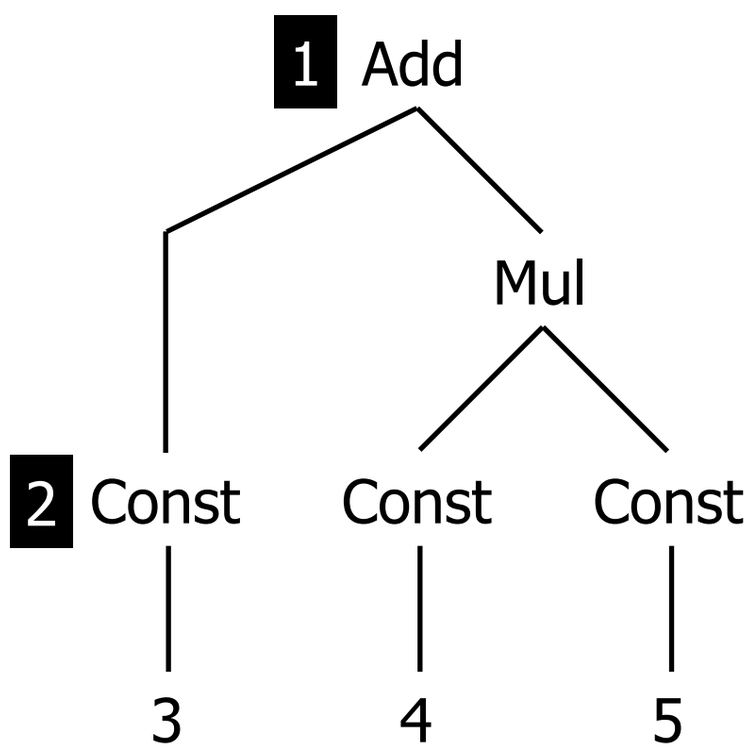
```
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

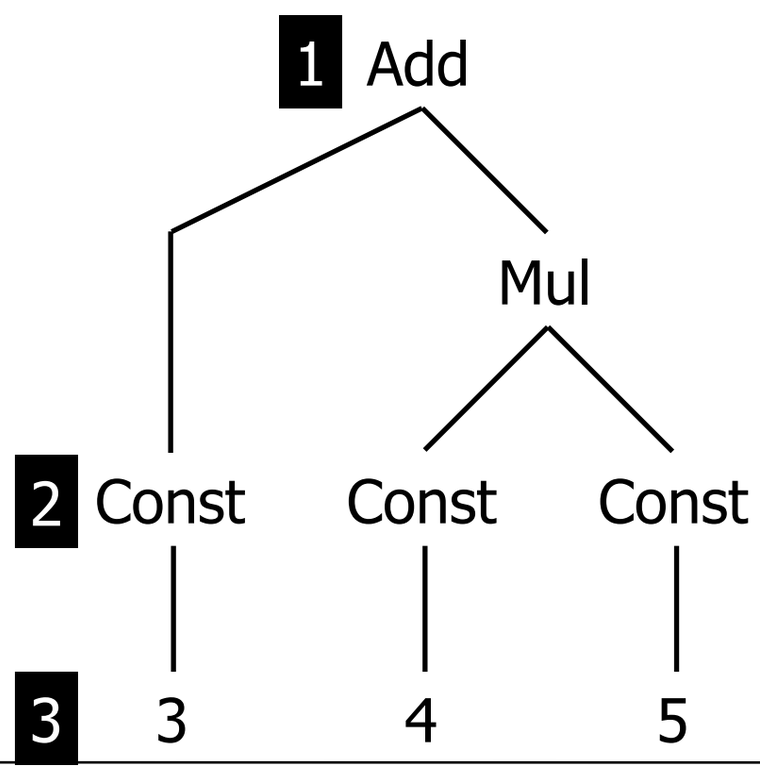
```
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

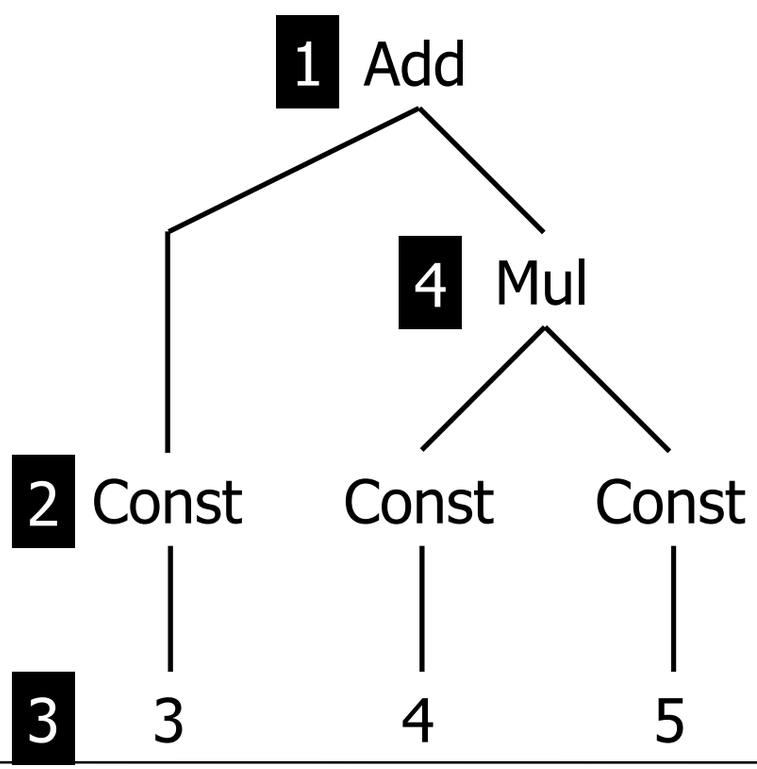
```
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

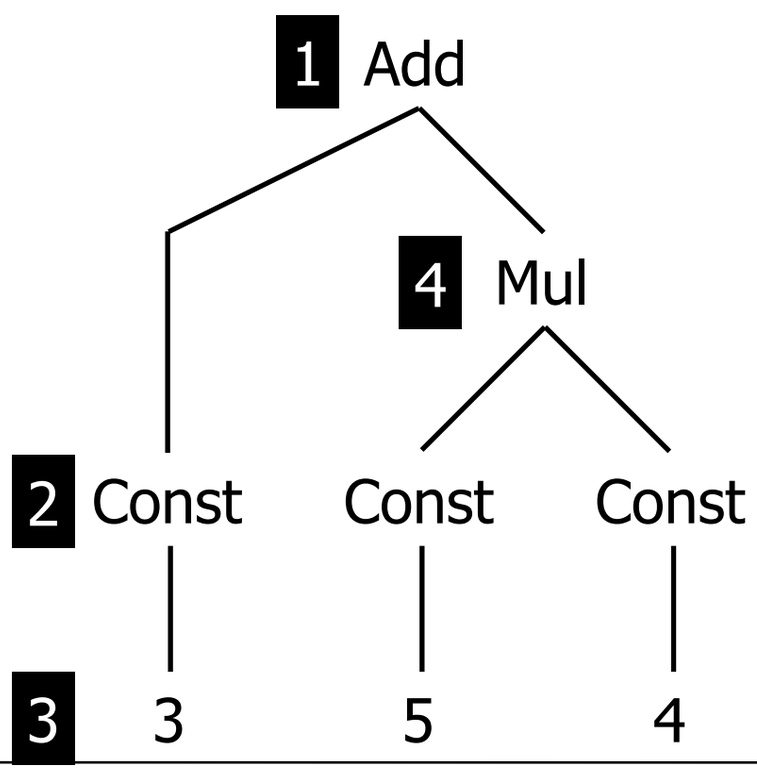
```
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

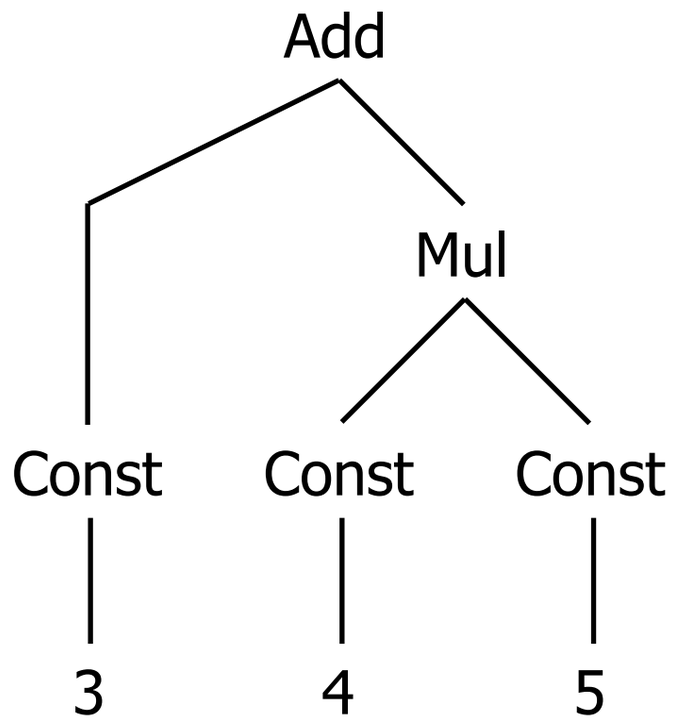
```
switch: Mul(e1, e2) -> Mul(e2, e1)

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

alltd(switch)
```



Stratego

example

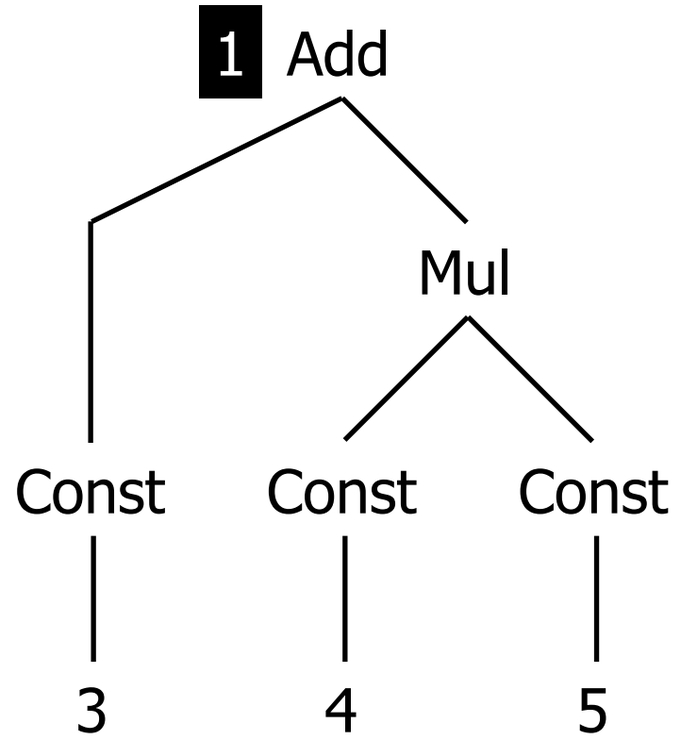
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

bottomup(switch)
```



Stratego

example

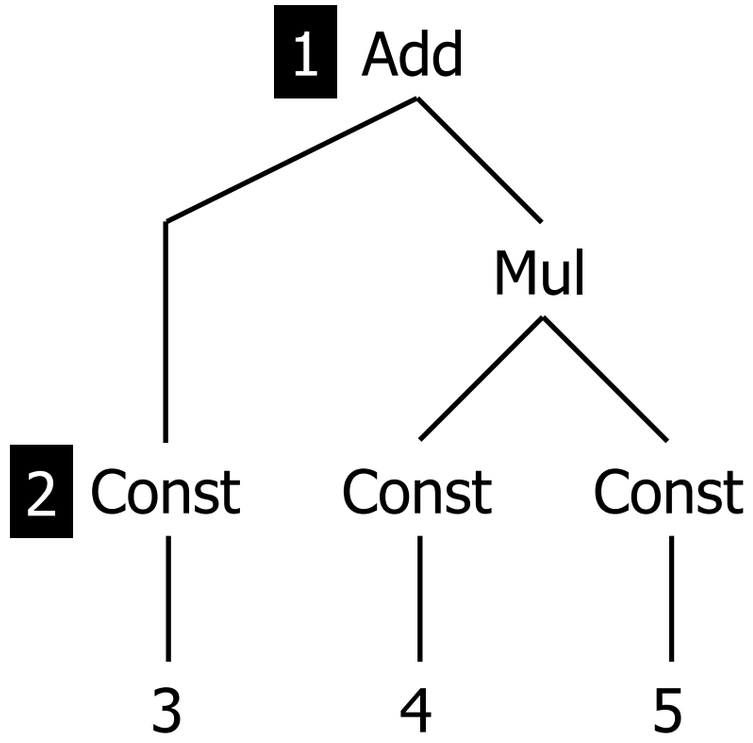
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

bottomup(switch)
```



Stratego

example

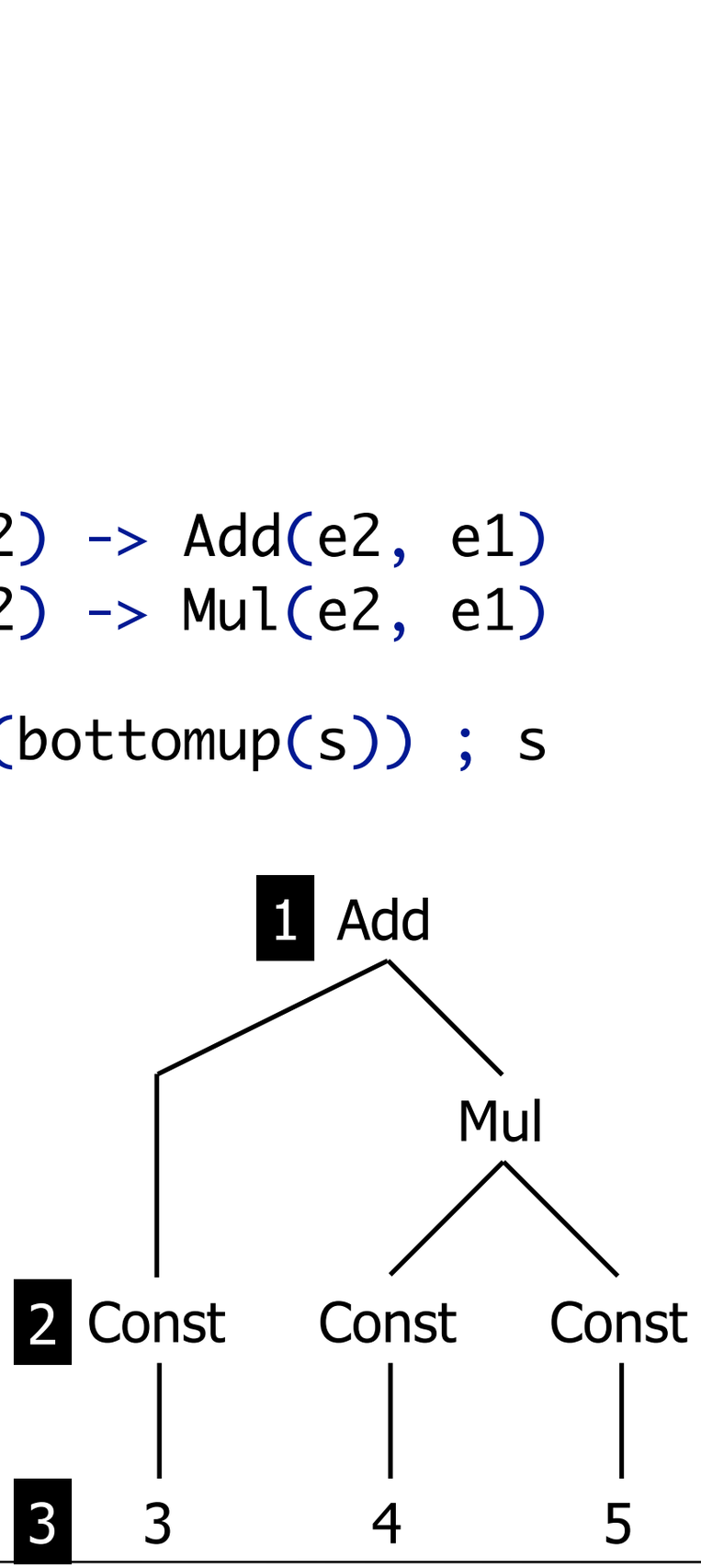
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

bottomup(switch)
```



Stratego

example

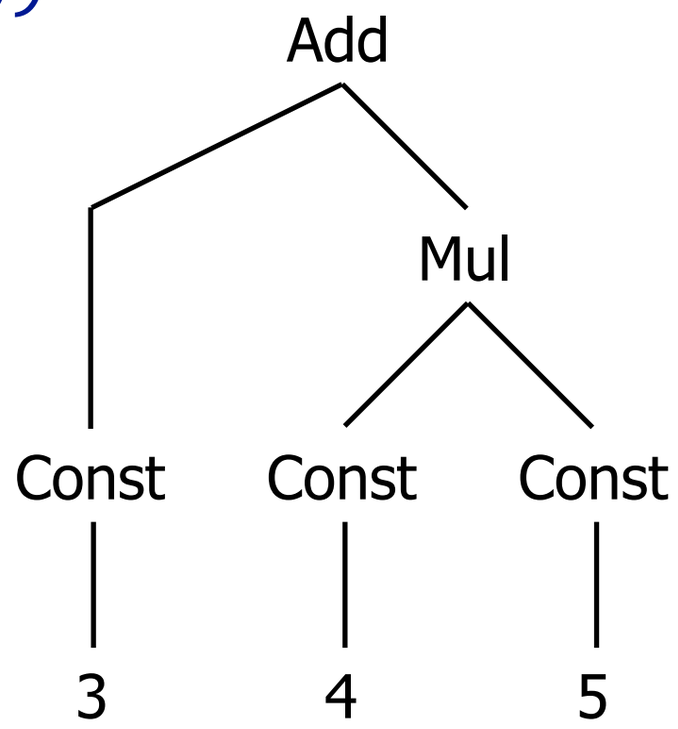
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

bottomup(switch)
```



Stratego

example

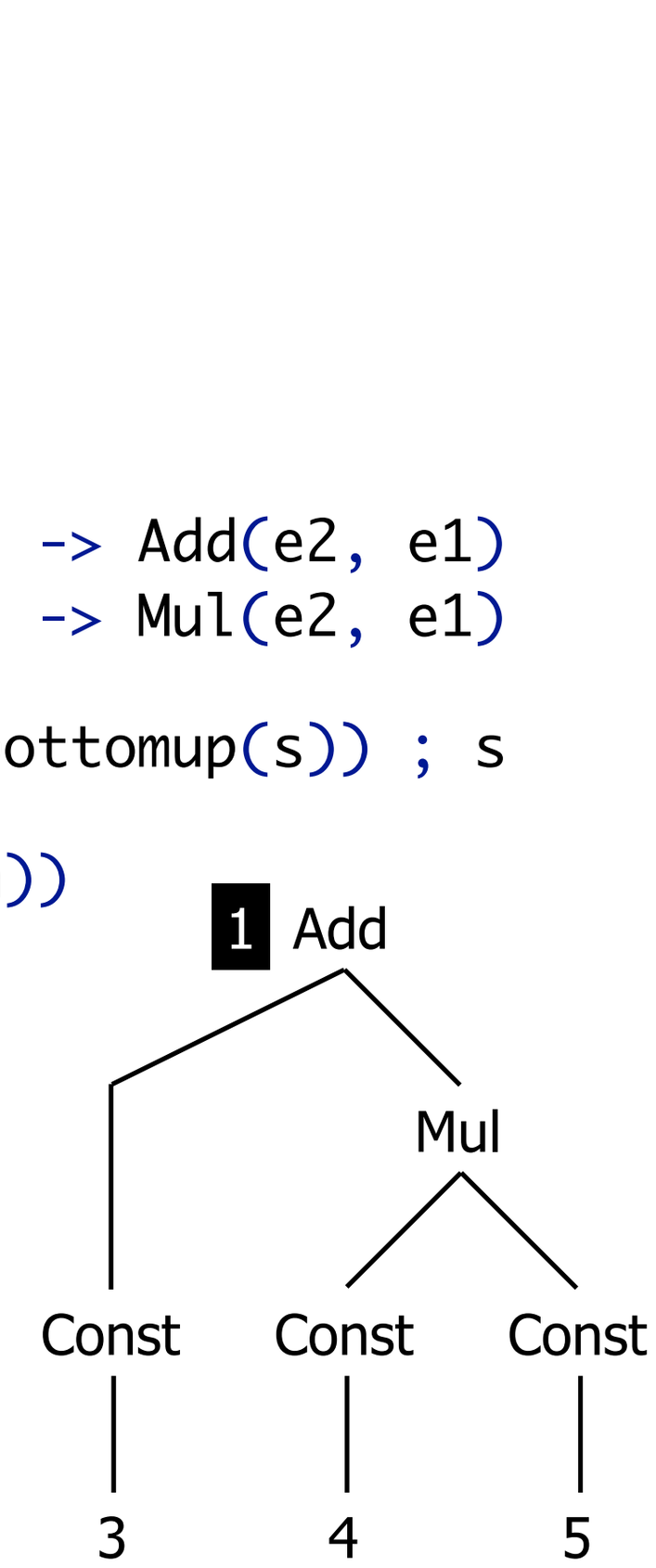
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

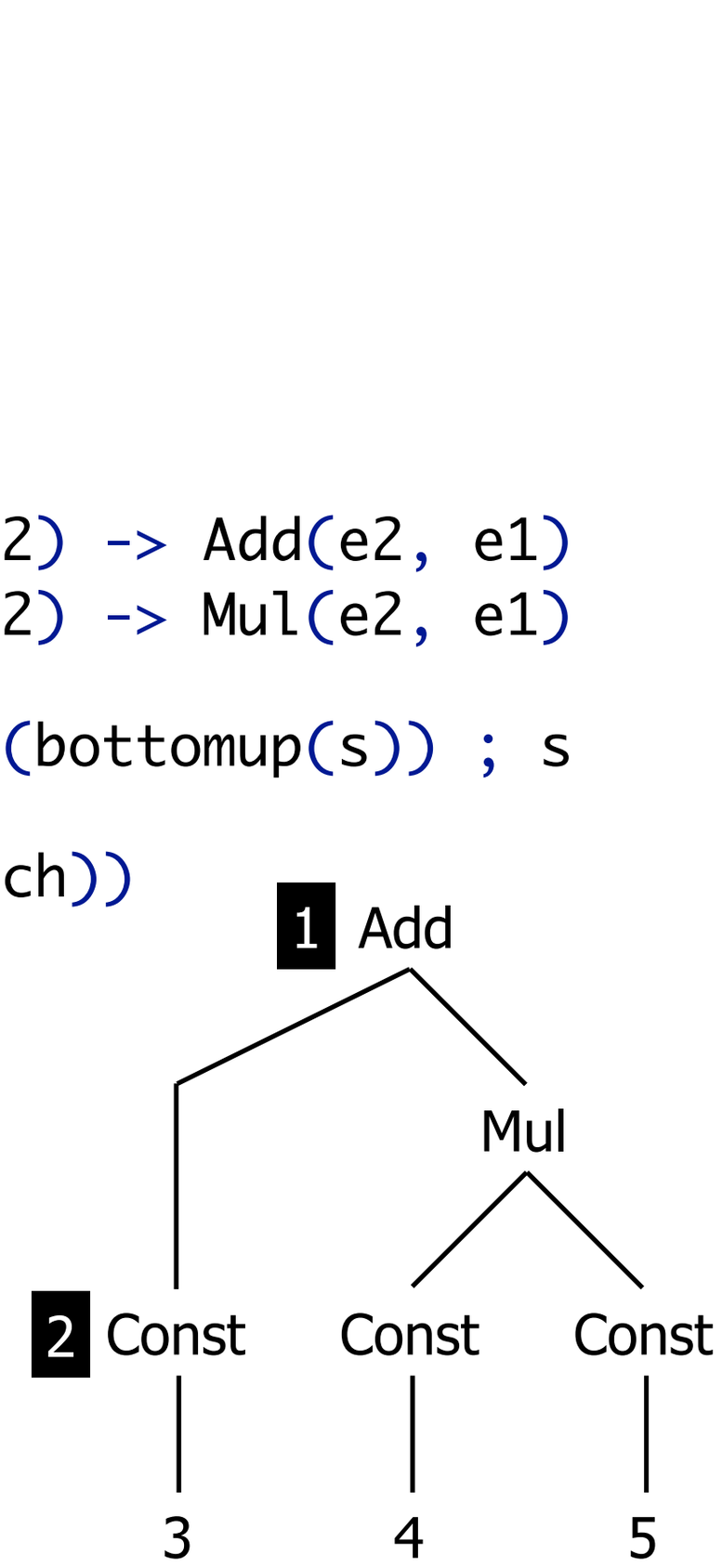
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

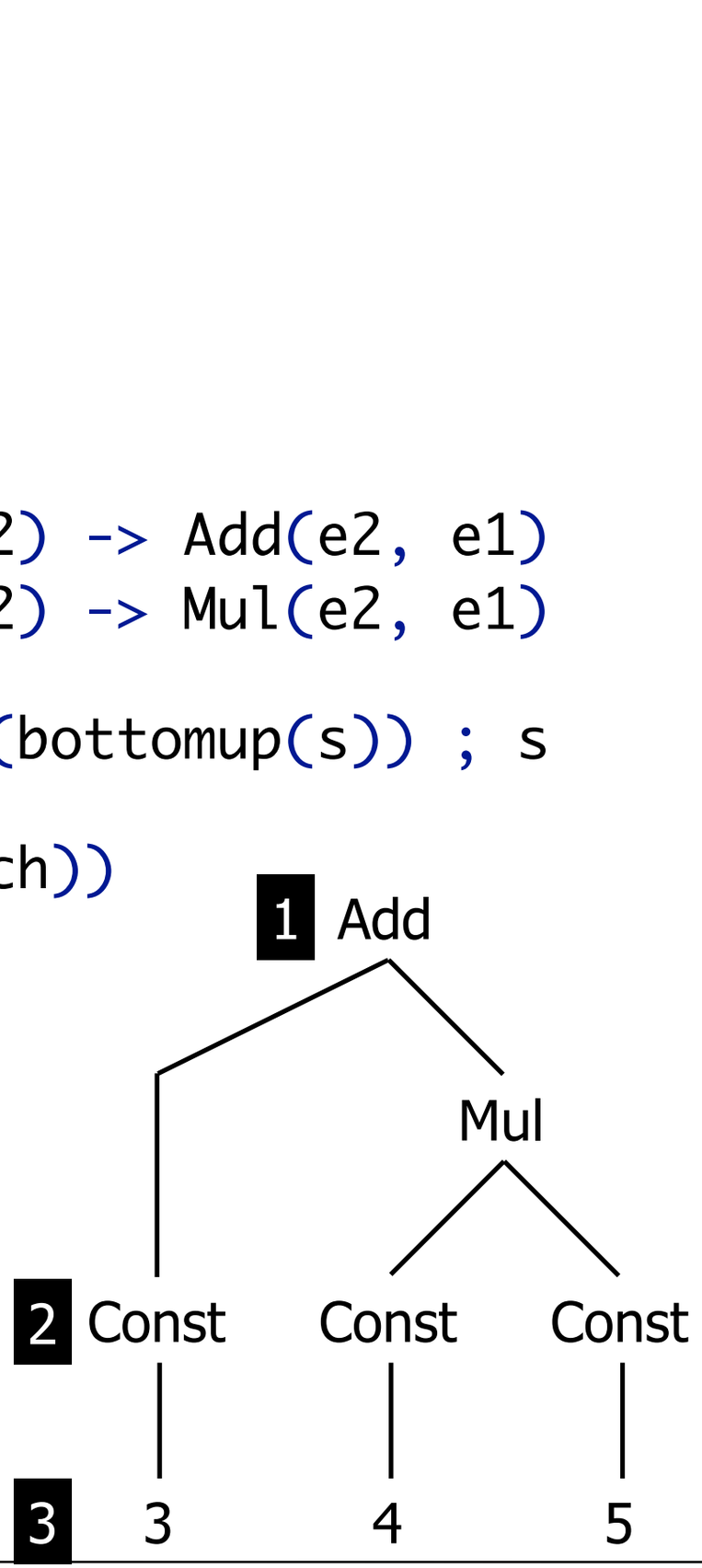
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

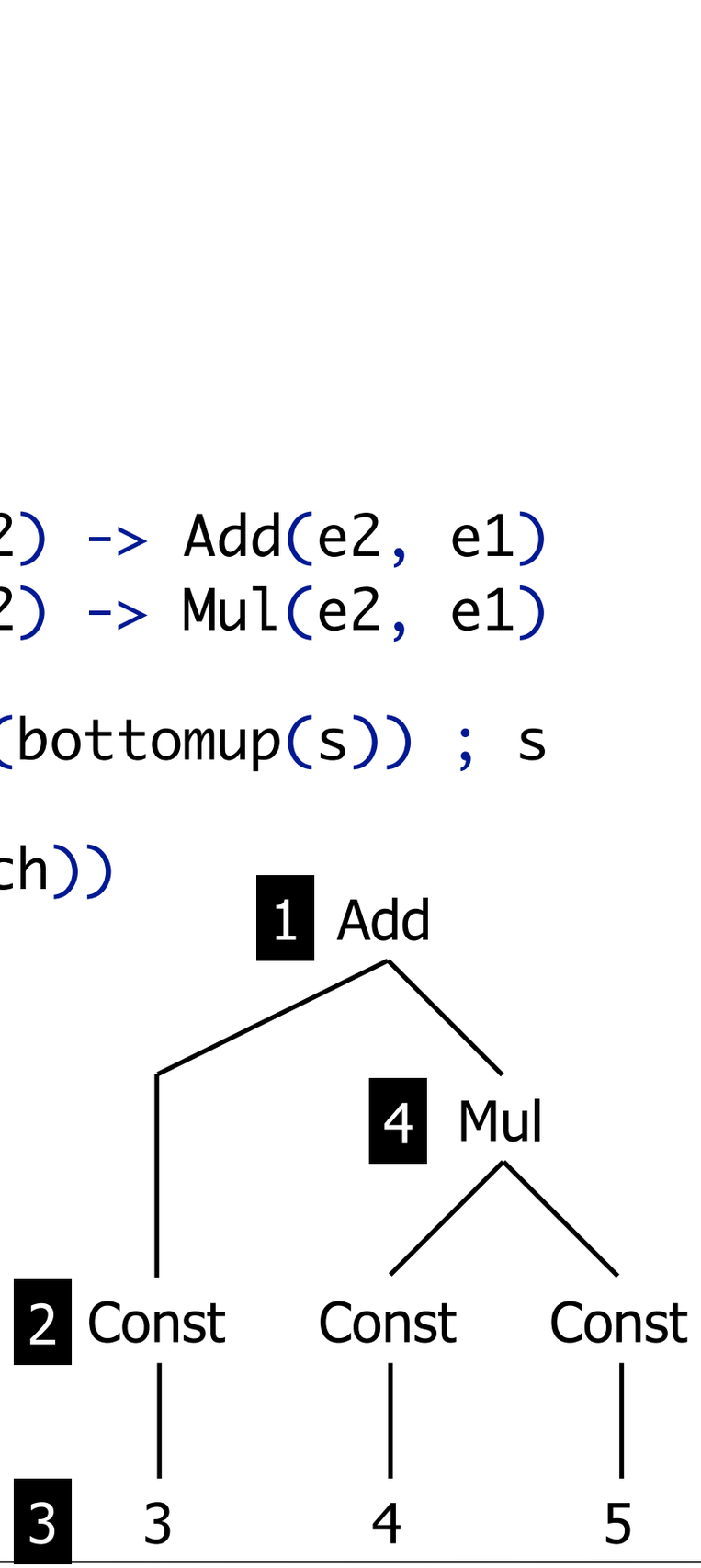
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

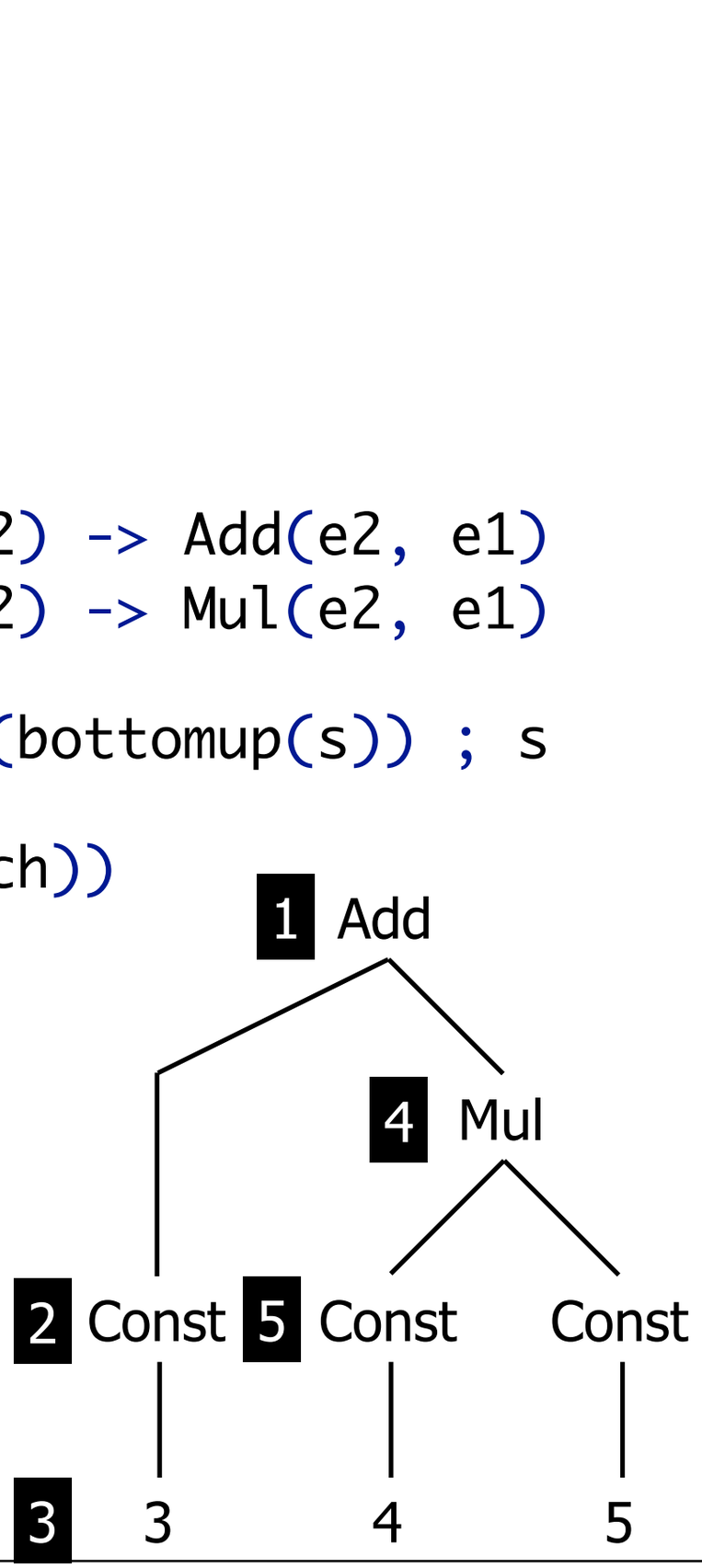
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

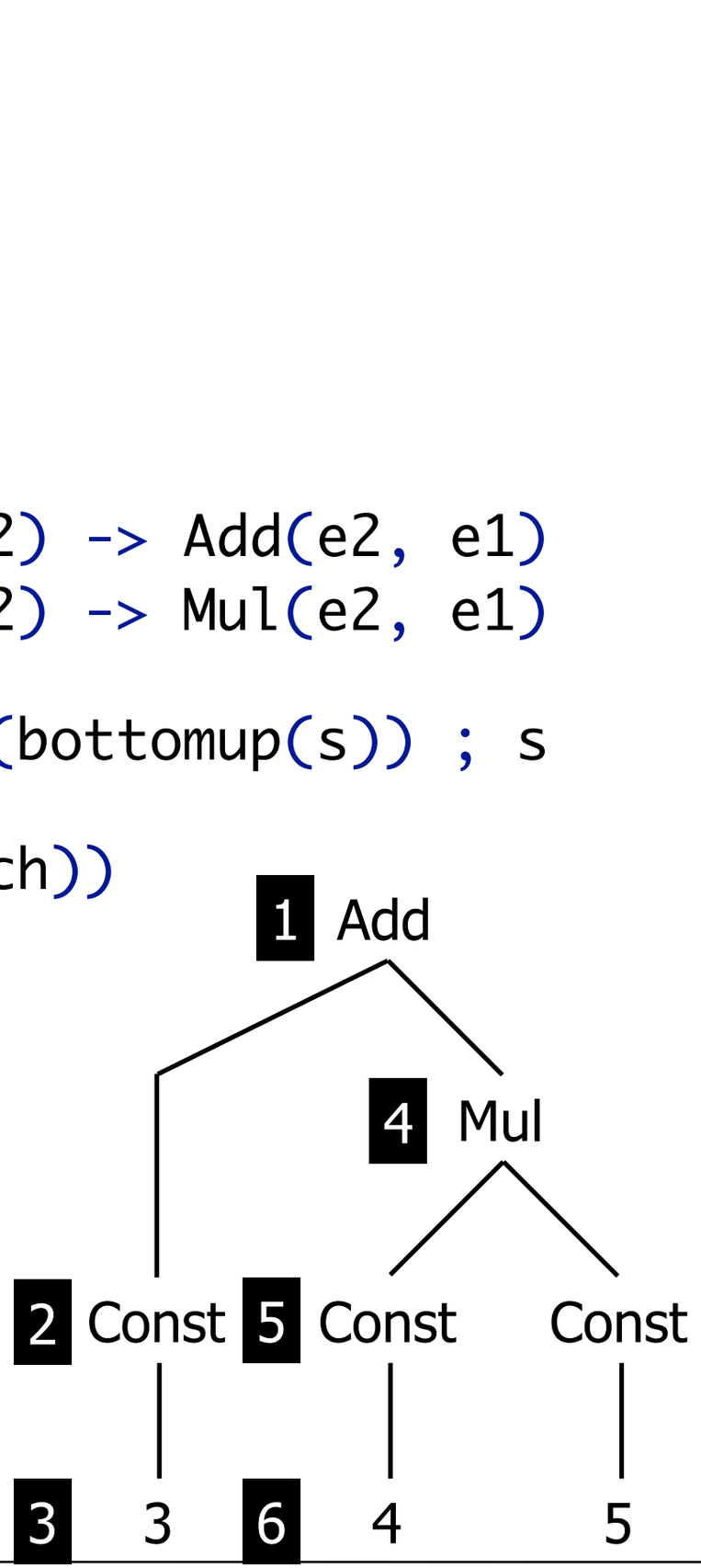
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

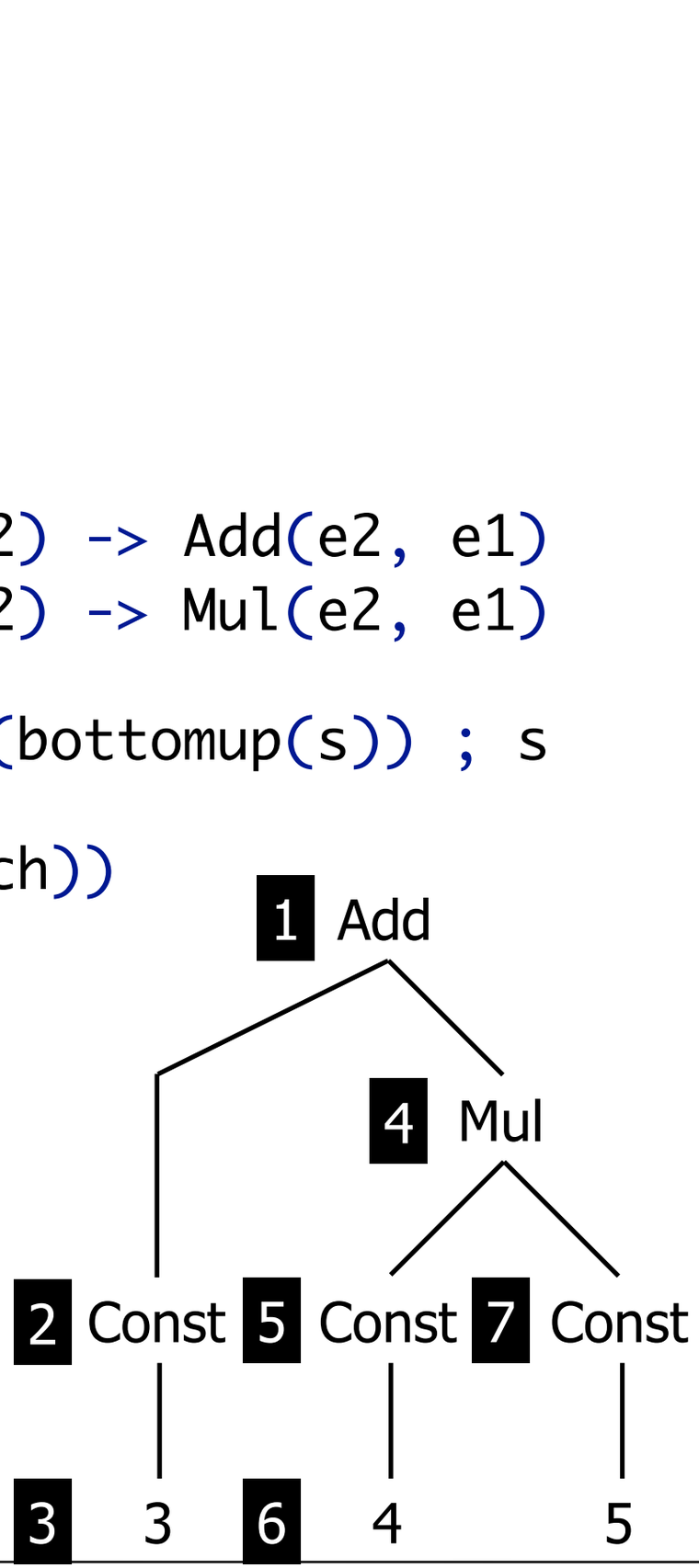
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

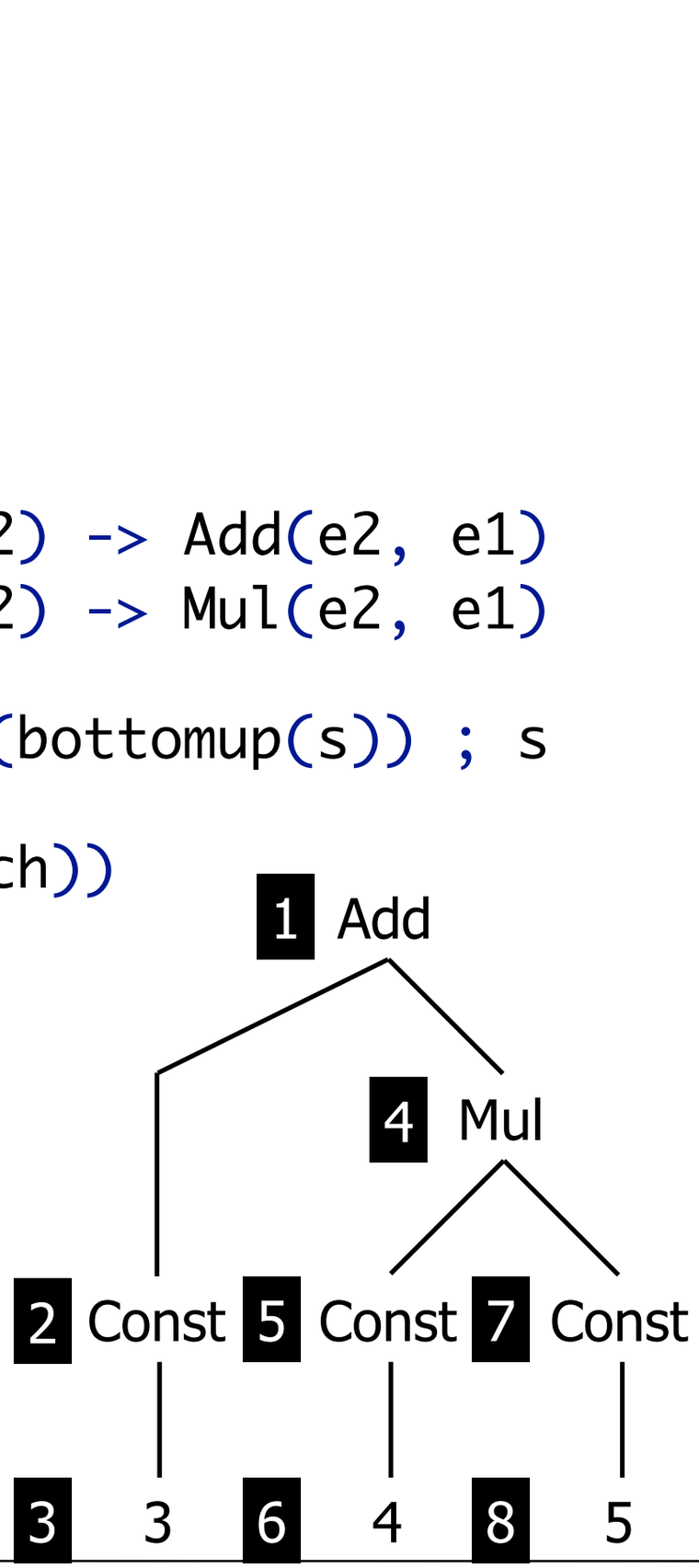
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

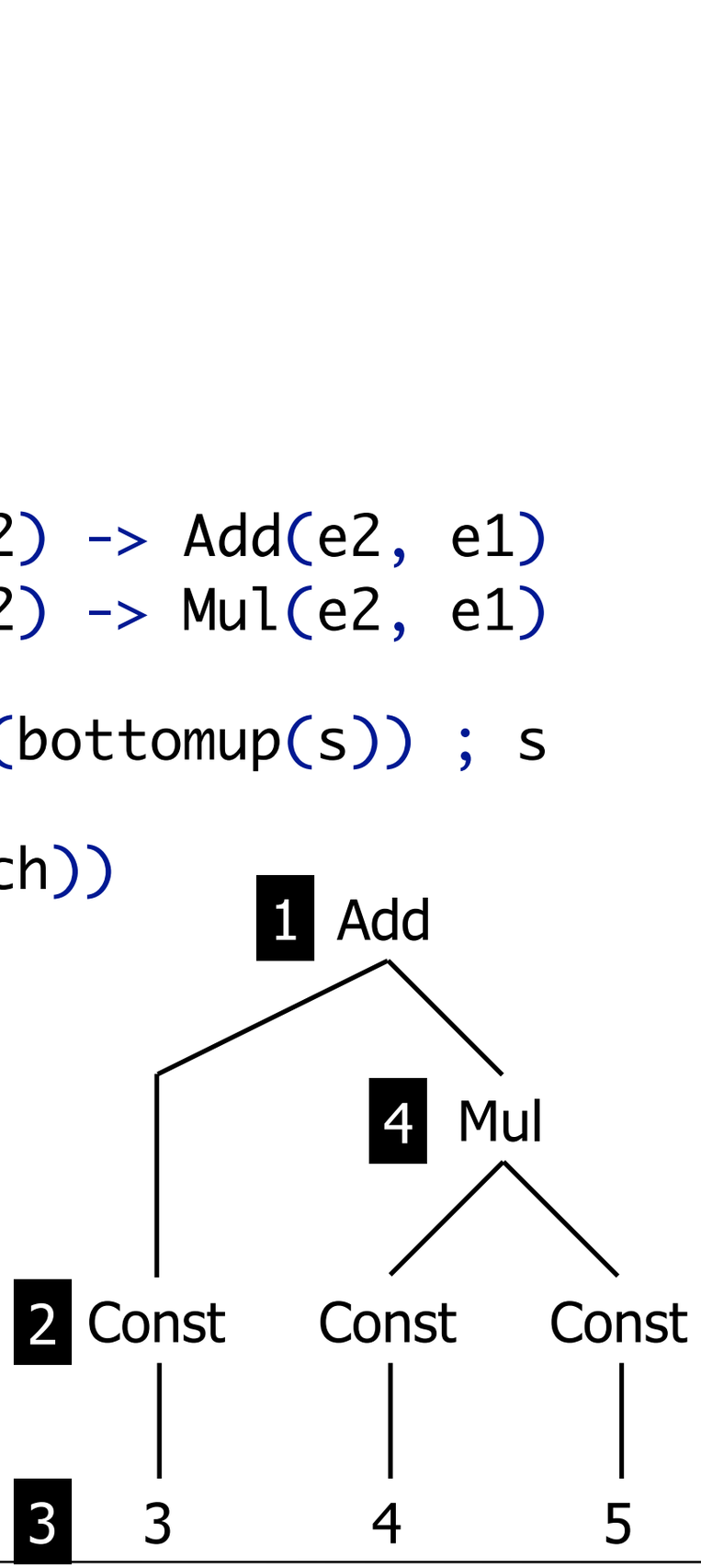
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

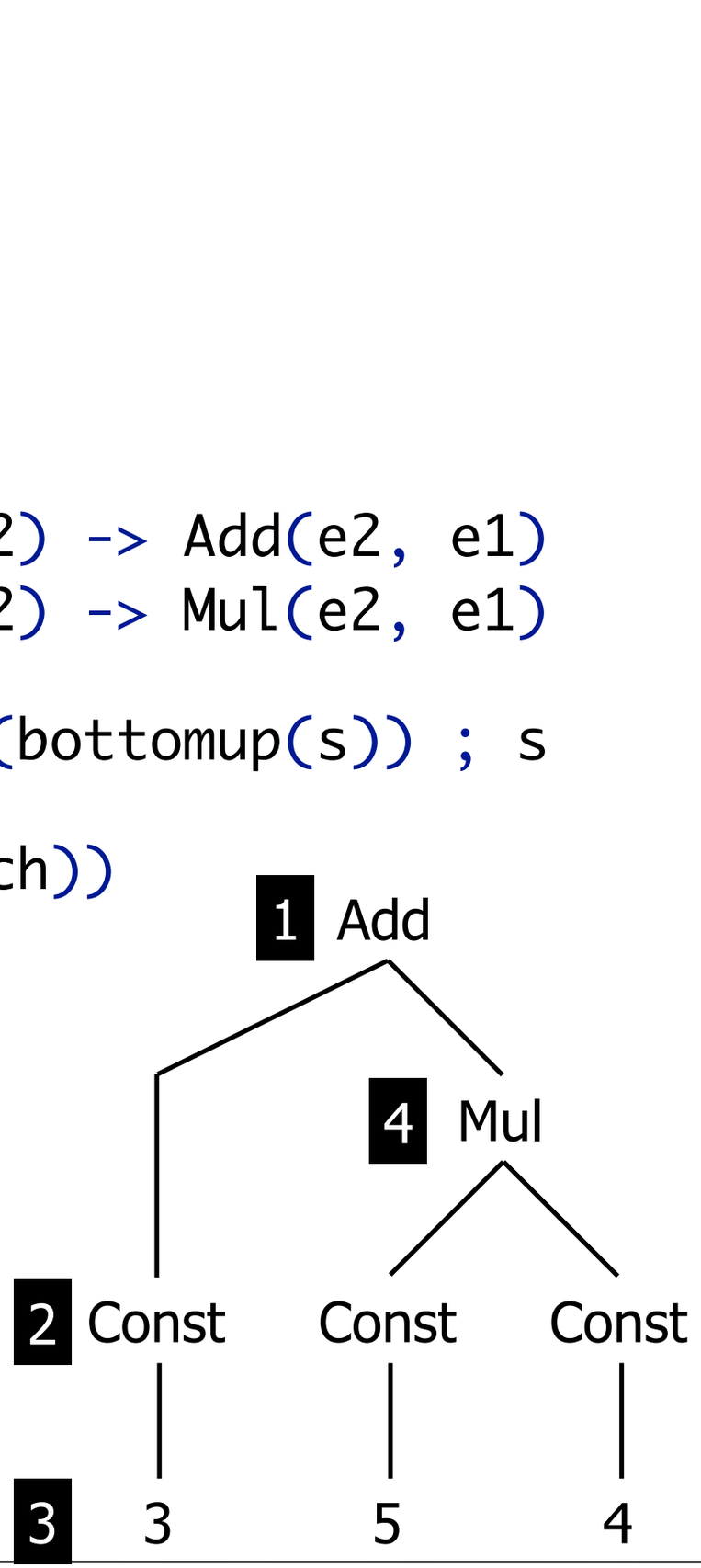
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

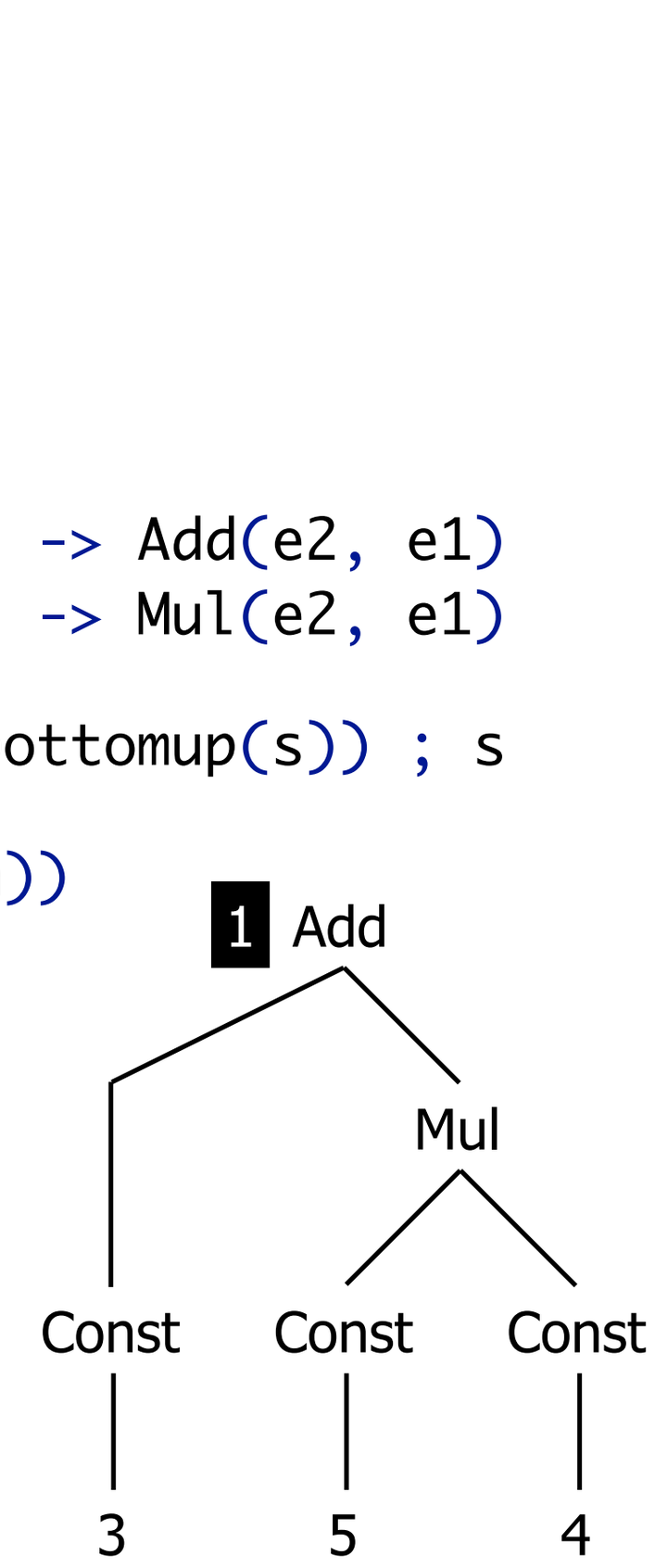
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

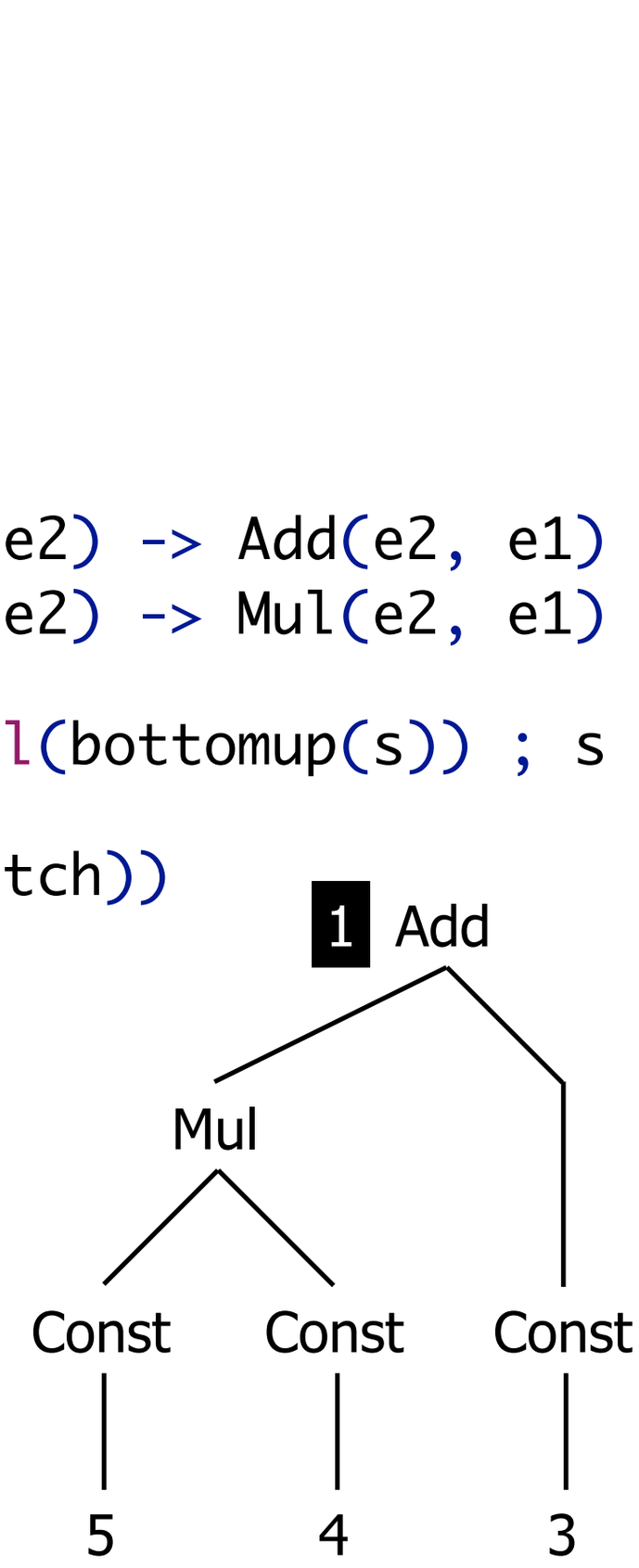
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

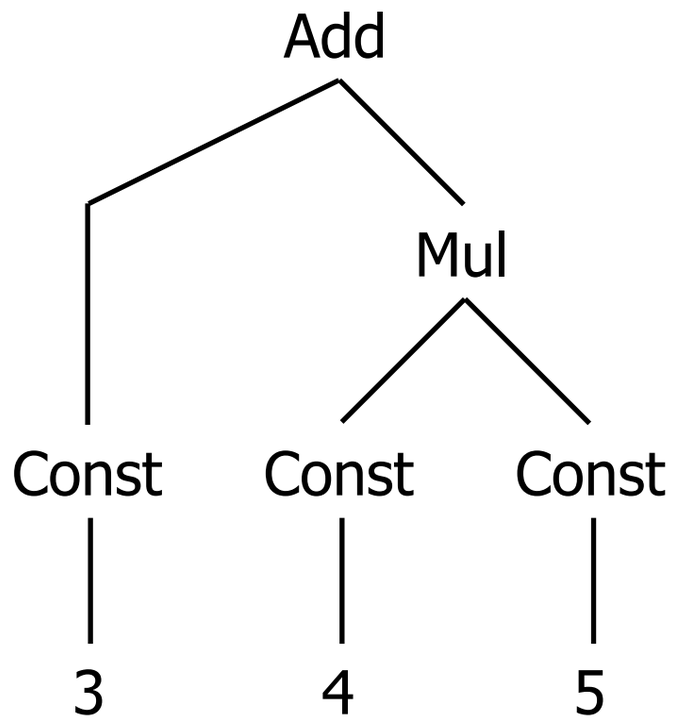
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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



Stratego

example

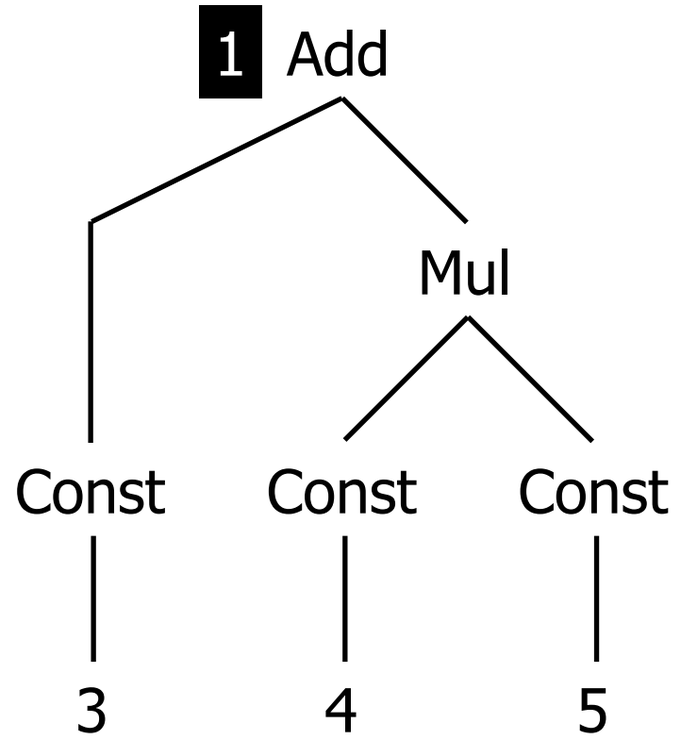
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

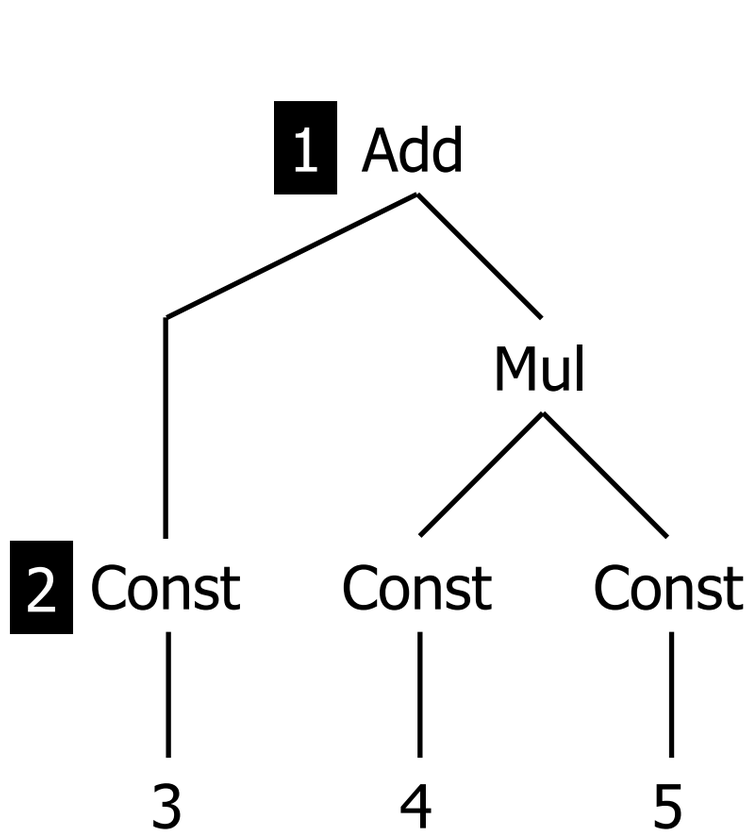
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

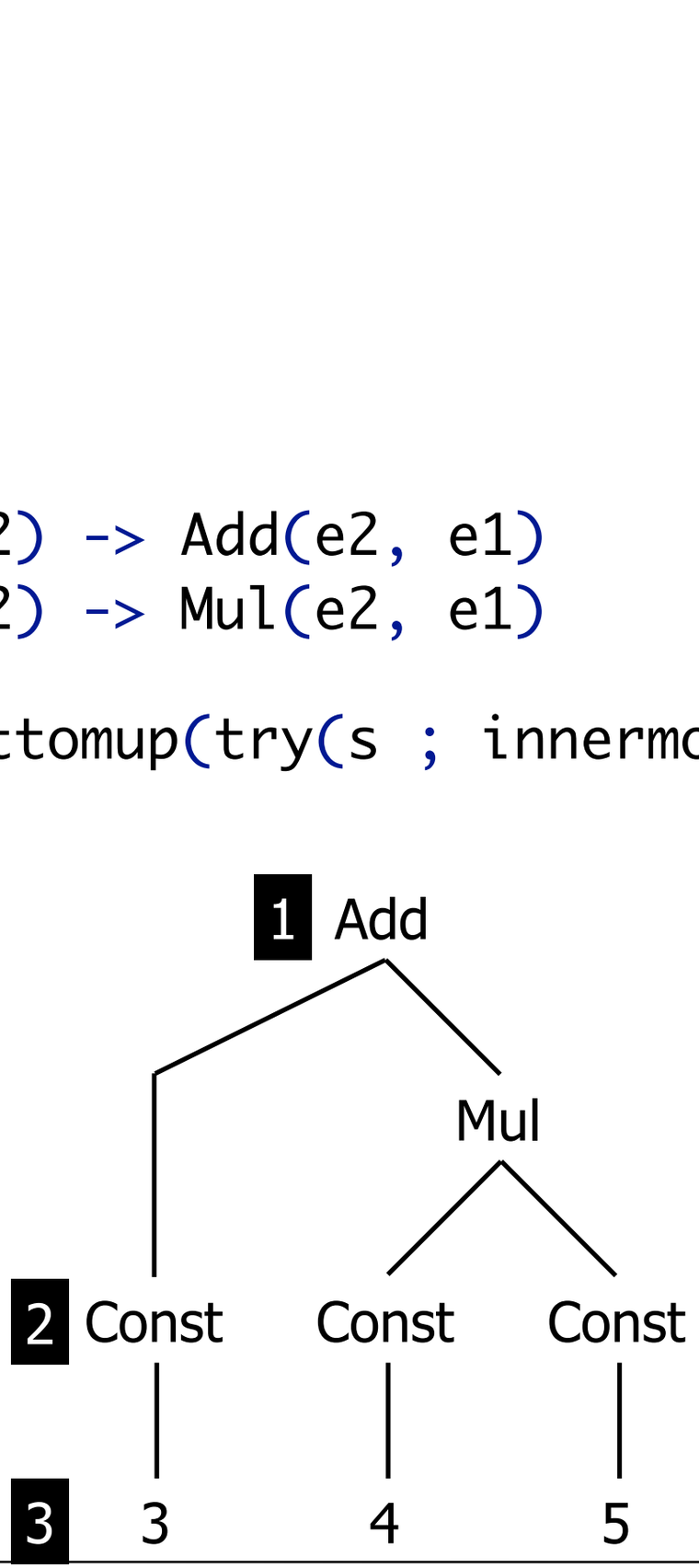
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

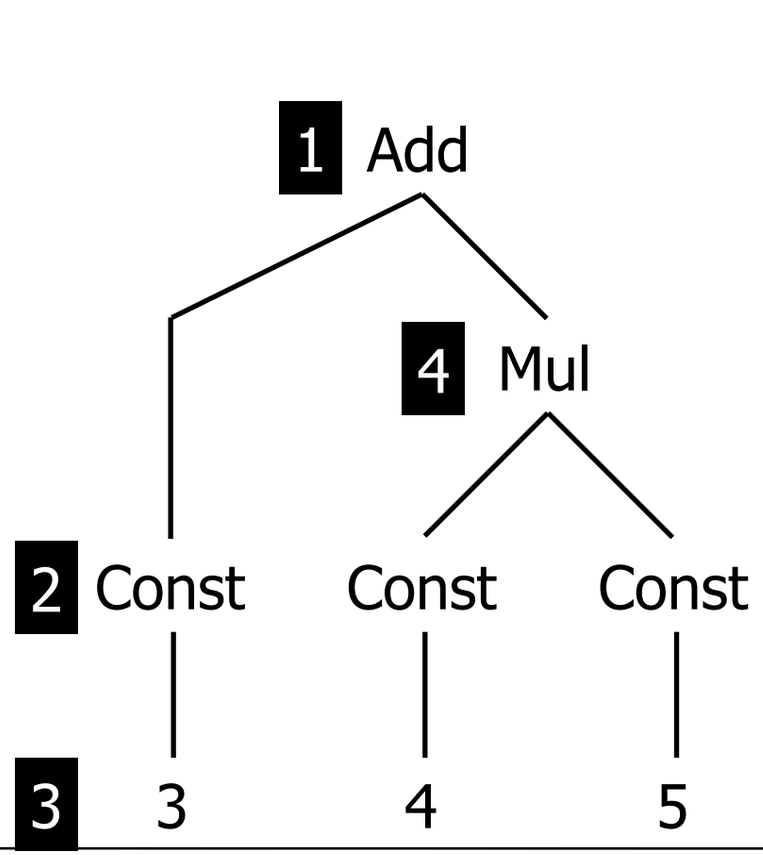
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

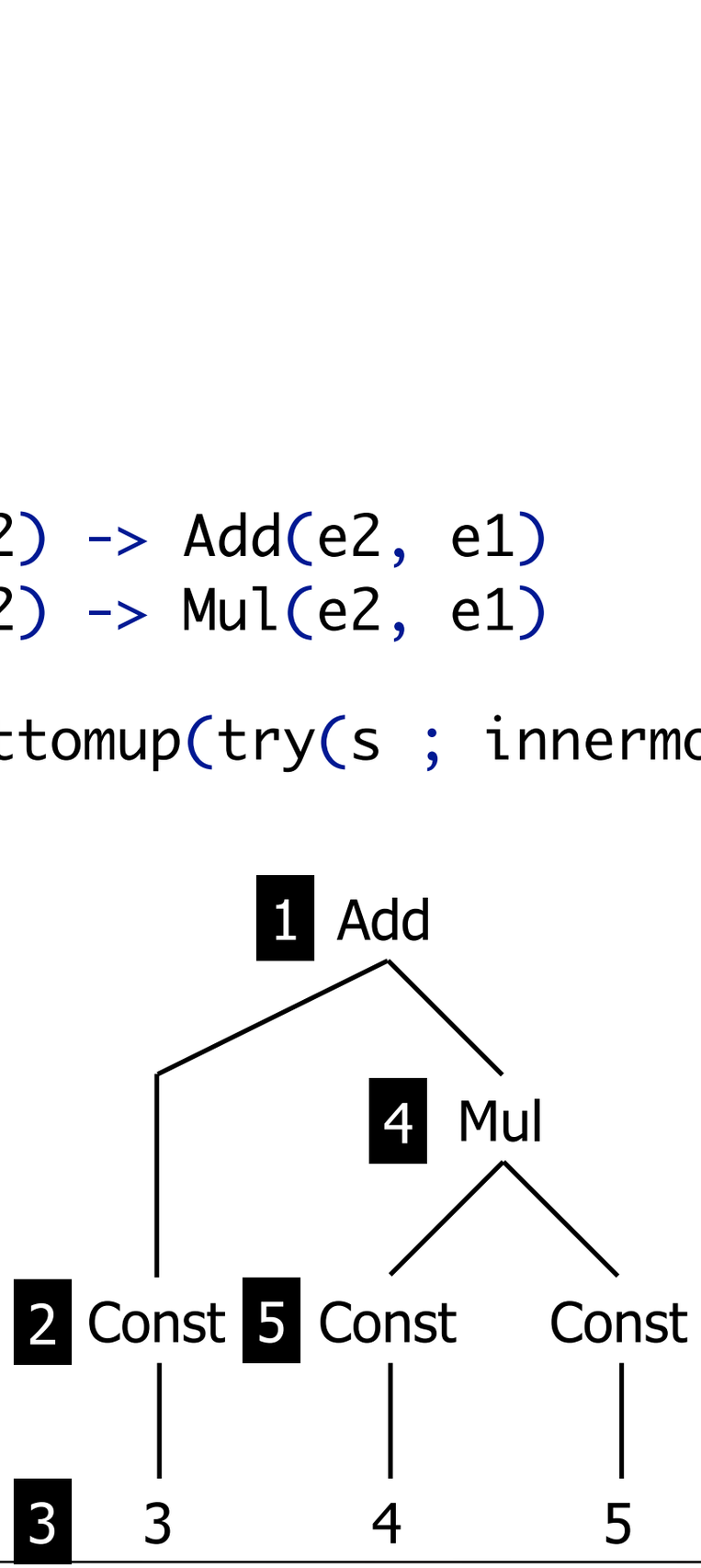
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

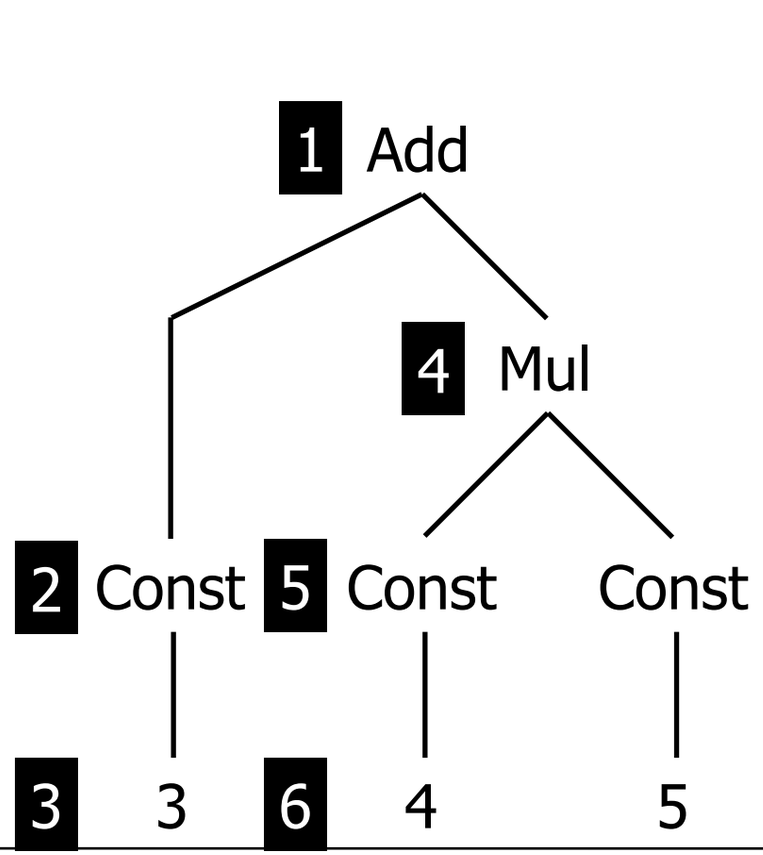
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

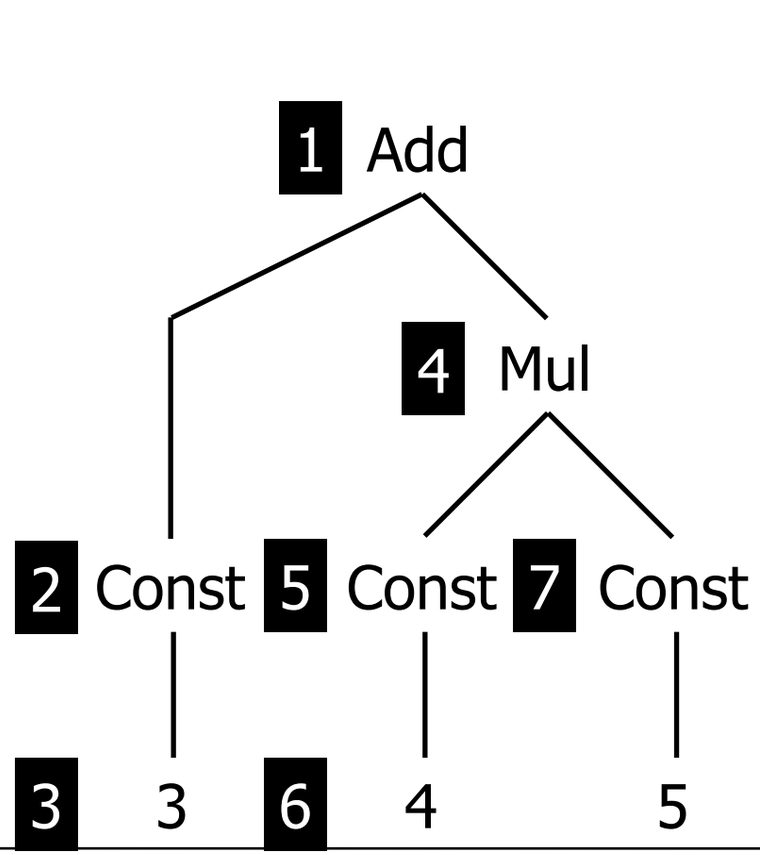
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

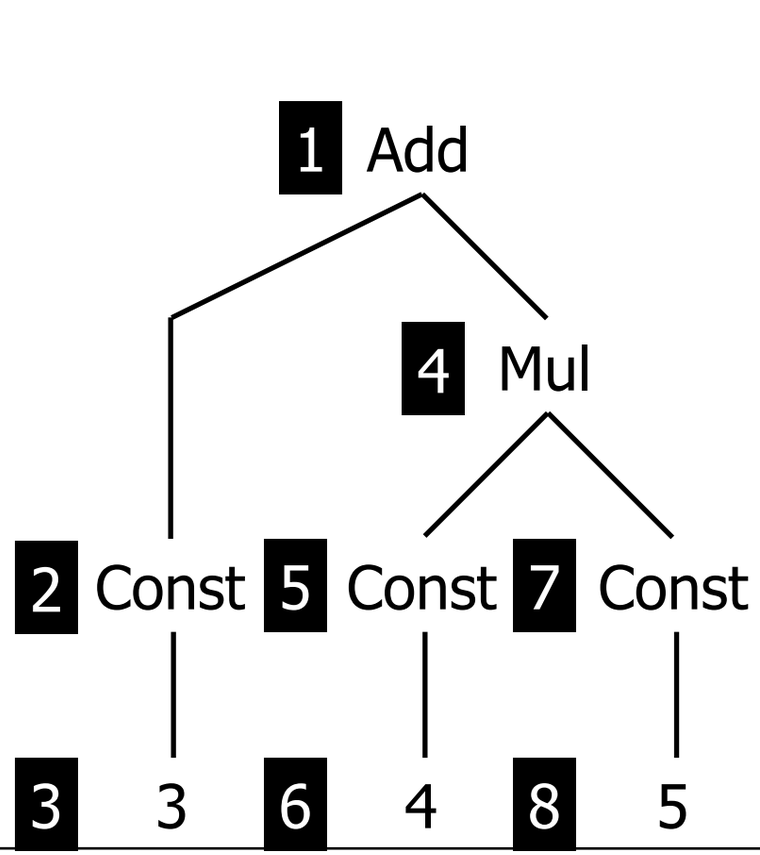
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

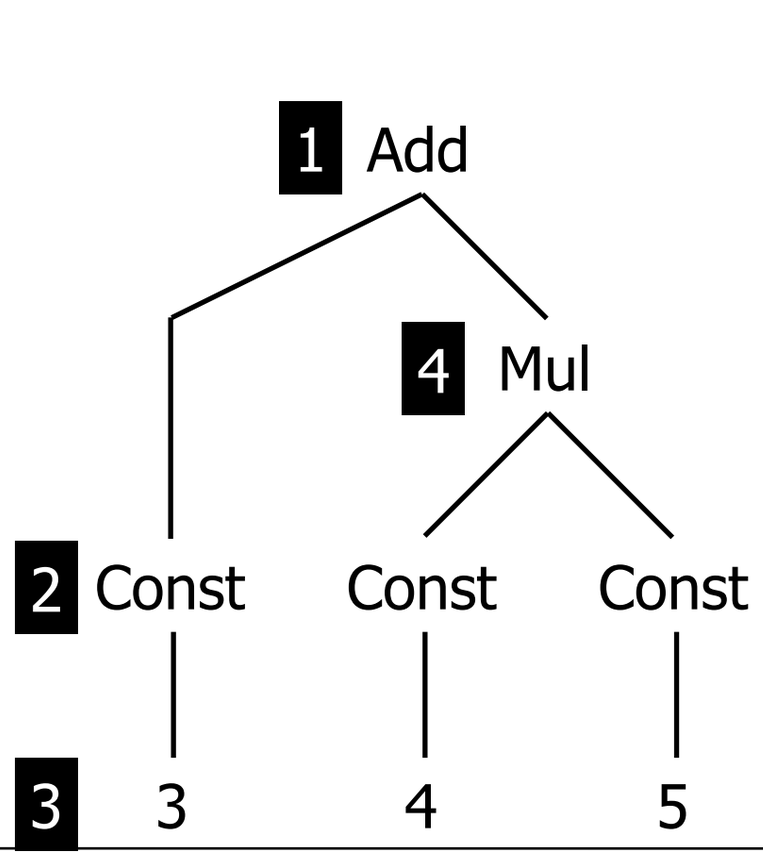
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

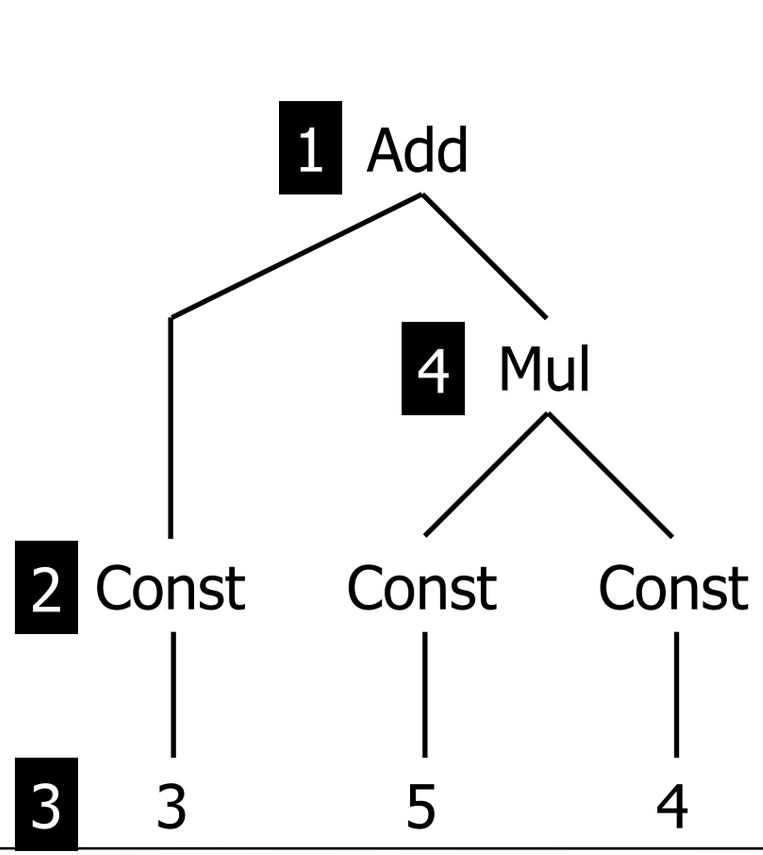
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

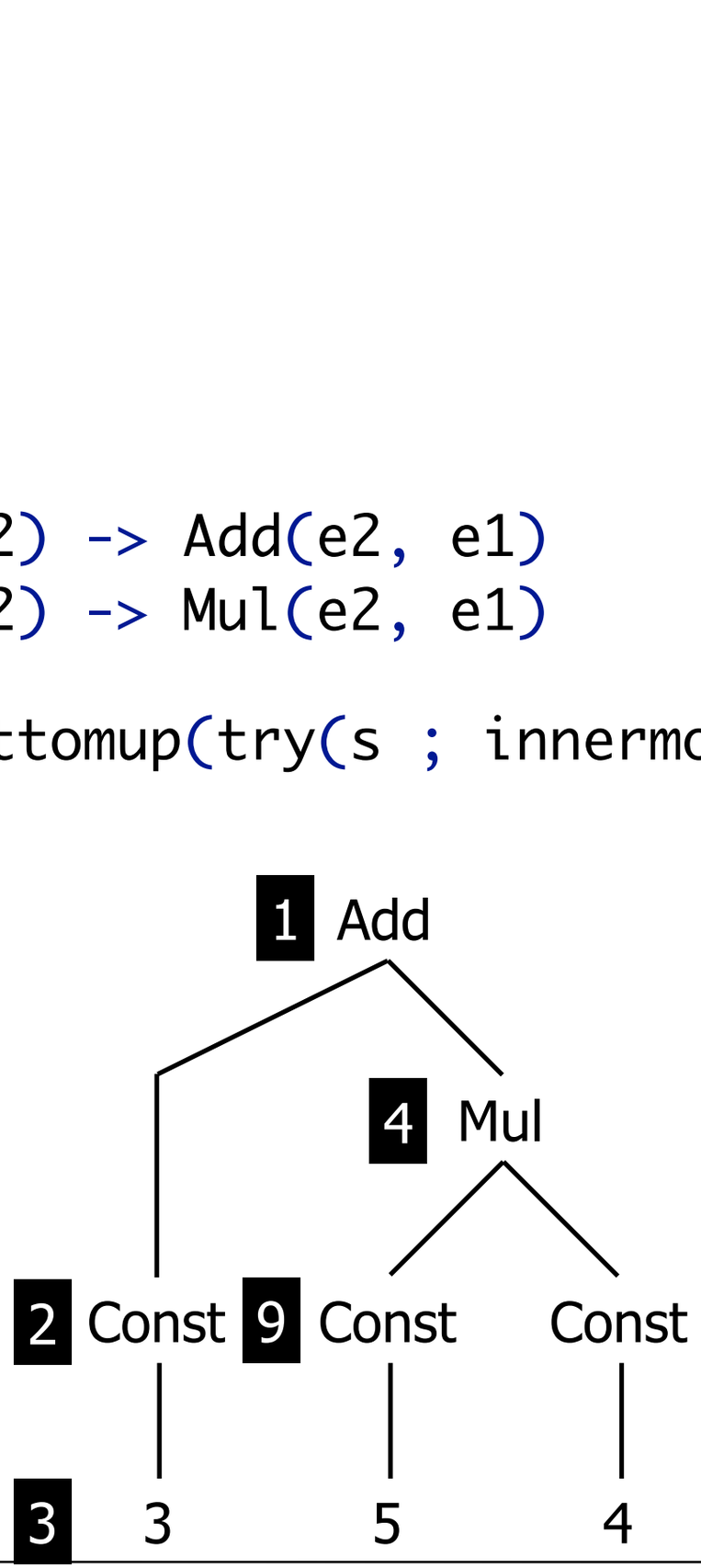
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

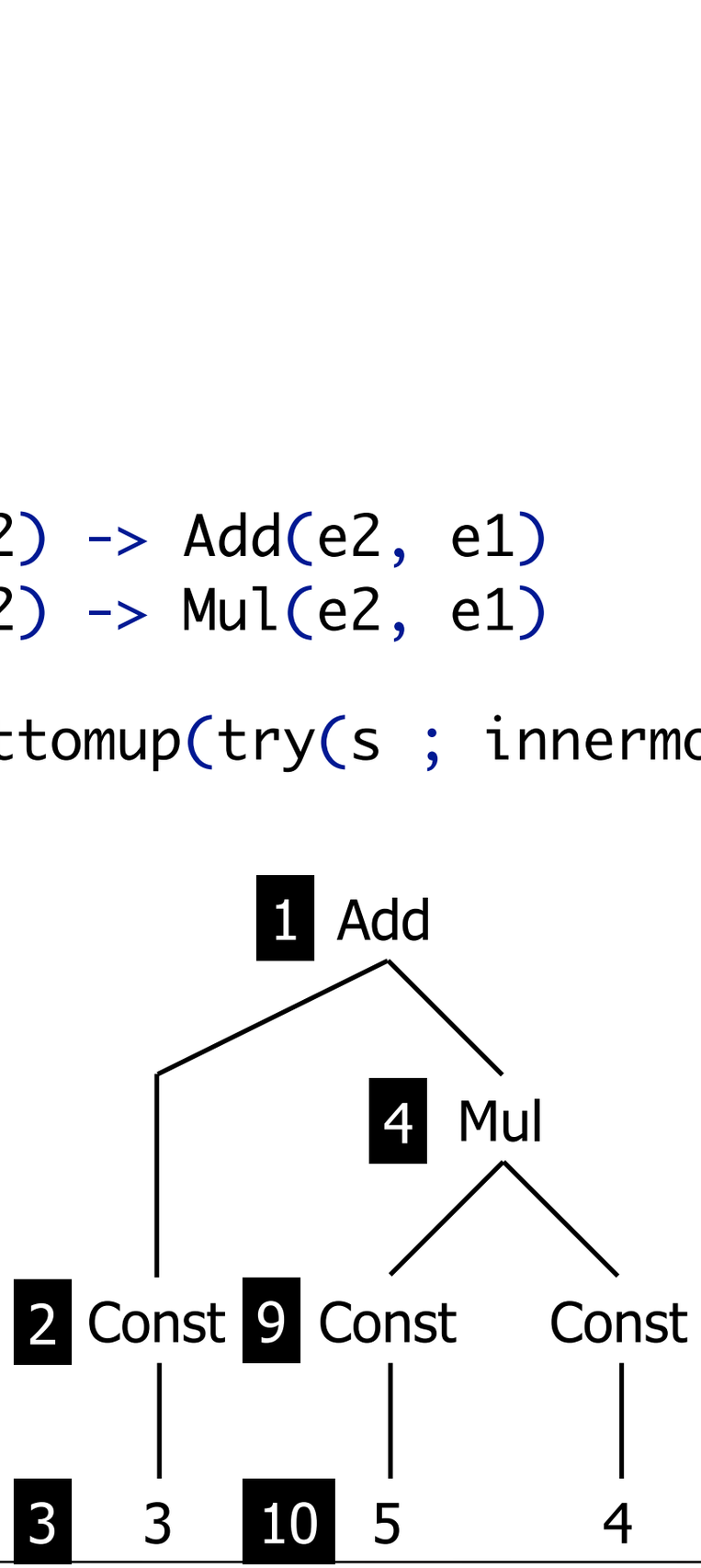
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

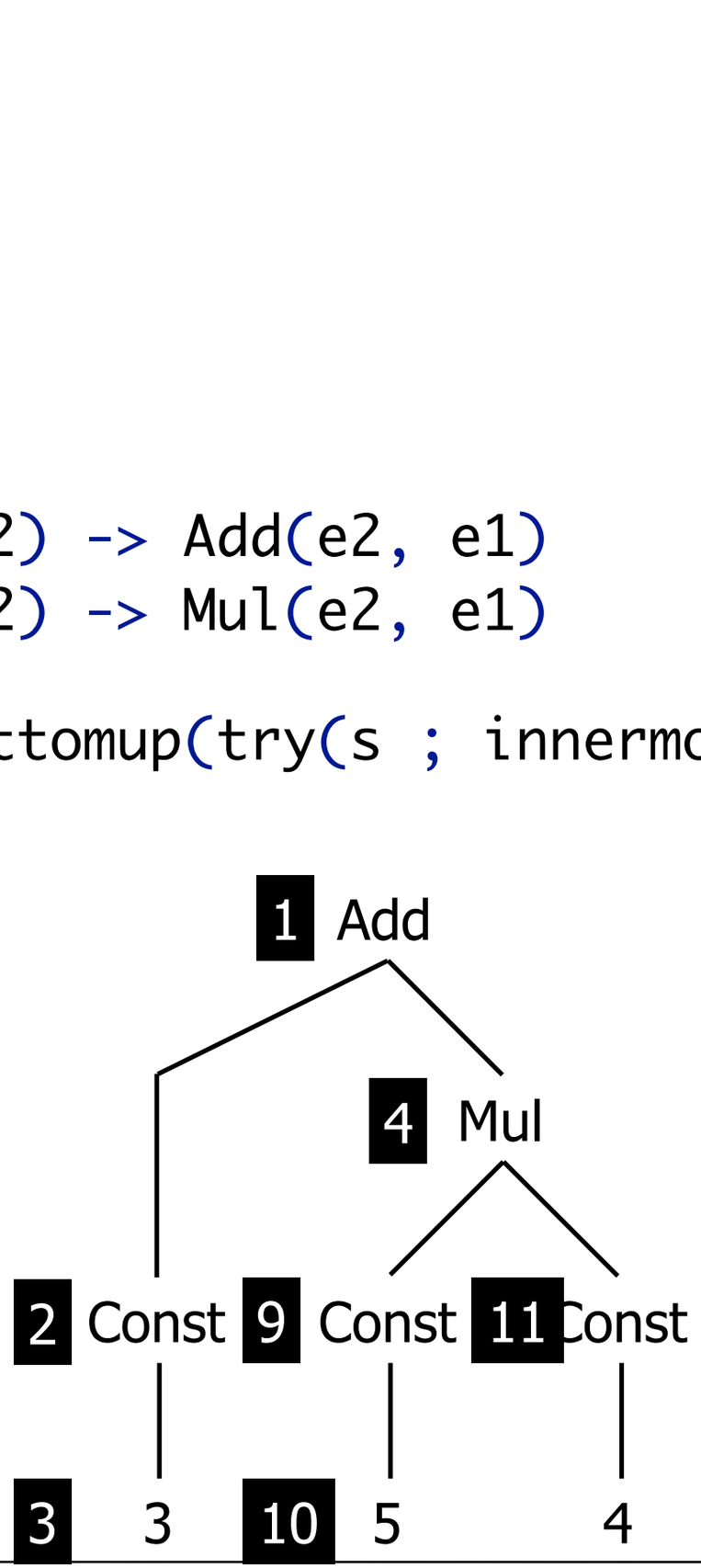
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

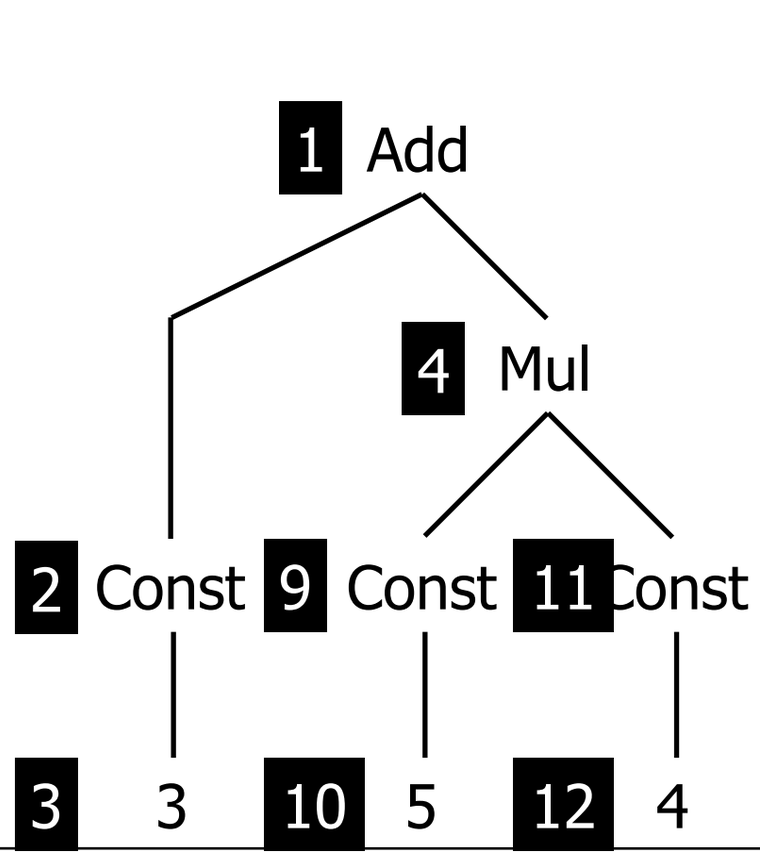
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

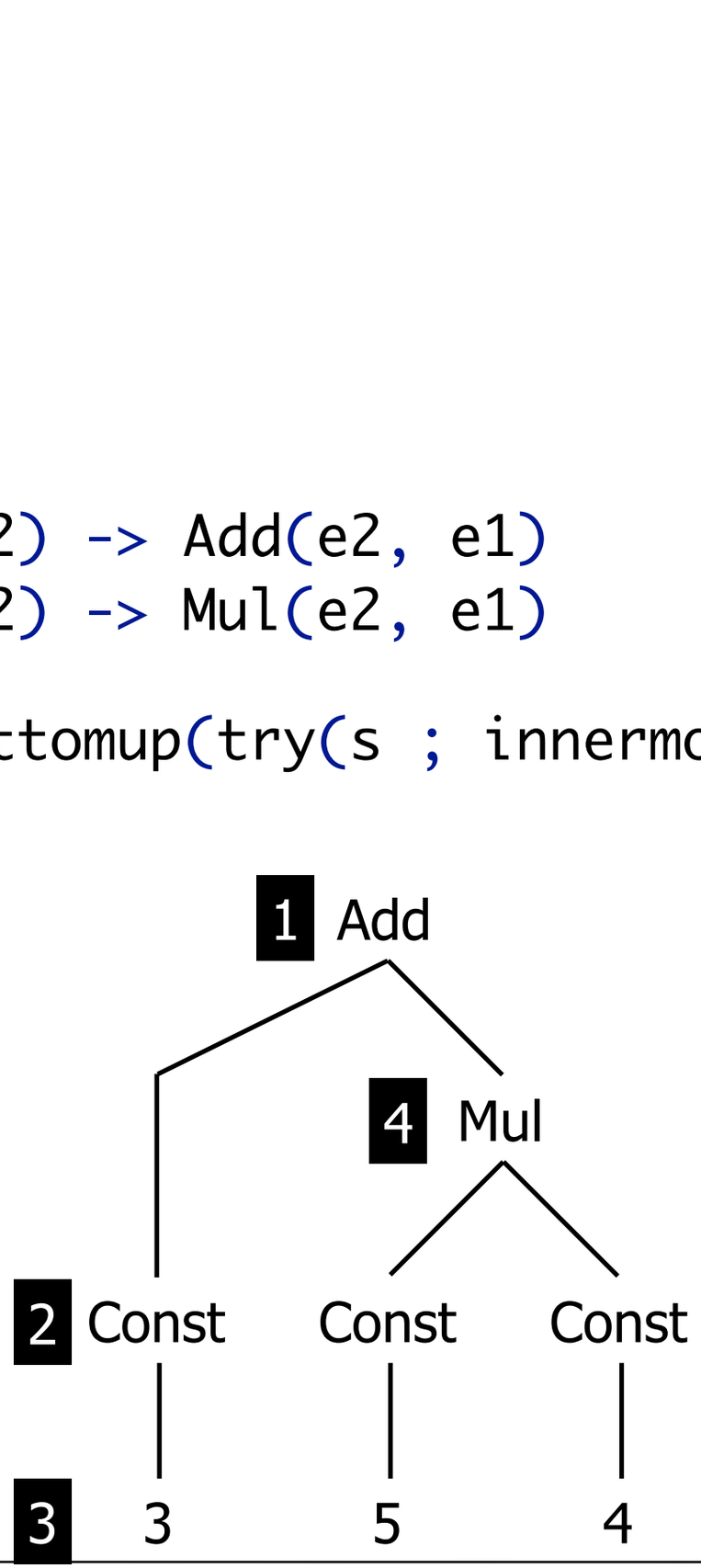
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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

innermost(switch)
```



Stratego

example

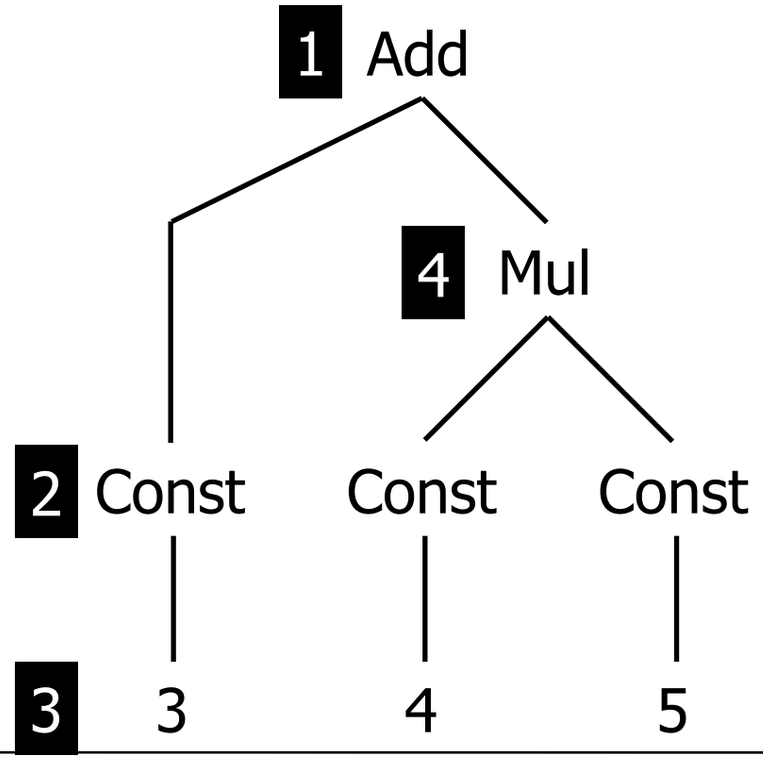
```
switch: Add(e1, e2) -> Add(e2, e1)
switch: Mul(e1, e2) -> Mul(e2, e1)

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)
```



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

attribution

slide	title	author	license
1, 35	Programming language textbooks	K.lee	public domain
2	The Bigger Picture	F. Delventhal	CC BY 2.0
5	Rose 2 des Kleinen Prinzen	Antoine Saint-Exupéry	public domain
11	Dog and owner on Ballykinler Beach	Jonny Green	CC BY 2.0
12	Fashionable melange of English words (1)	trialsanderrors	public domain
13	Gift	asenat29	CC BY 2.0
13	Toxic	John Morgan	CC BY 2.0
14	Latin Grammar	Anthony Nelzin	
15	Ginkgo Biloba	Johann Wolfgang von Goethe	public domain
16	Sub-deb slang	genibee	CC BY-NC 2.0
17	Wednesday	Michael Fawcett	CC BY-NC-SA 2.0
19, 25, 67	Thesaurus	Enoch Lau	CC BY-SA 3.0
22	The captured Swiftsure, Seven Oaks, Loyal George and Convertine brought through Goeree Gat	Willem van de Velde the Younger	public domain

Stra

slide	title	author	license
23	Tower of Babel	Pieter Bruegel the Elder	public domain
26	The Ashtadhyayi	translated by Srisa Chandra Vasu	public domain
27	Buchdruckerduden		public domain
28	Boeing 737-300 -400 -500 Manitenance Manual Alaska Airlines	Bill Abbott	CC BY-SA 2.0
30	Esperanto		public domain
31	Quenya	Juanma Pérez Rabasco	CC BY 2.0
32	Klingon Dictionary	Josh Bancroft	CC BY-NC 2.0
33	Overweight Na'vi	HARRY NGUYEN	CC BY 2.0
36, 45, 46	IBM Electronic Data Processing Machine	NASA	public domain
41	Tiger	Bernard Landgraf	CC BY-SA 3.0
42	The C Programming Language	Bill Bradford	CC BY 2.0
43	Italian Java book cover		
49-73	PICOL icons	Melih Bilgil	CC BY 3.0