Interpreted Pattern Match Execution

Jeff Niu

Agenda

- Background and motivation
- Turning patterns into bytecode
- Dialect for patterns
- Optimizing patterns
- Dialect for a state machine
- Quick look at the interpreter
- Performance metrics
- Current status and what's next

Background

Match and Rewrite

- Underlies many MLIR code transformations
- Not runtime extensible
- Has room for match optimization
- Binary size bloat

Runtime Extensibility

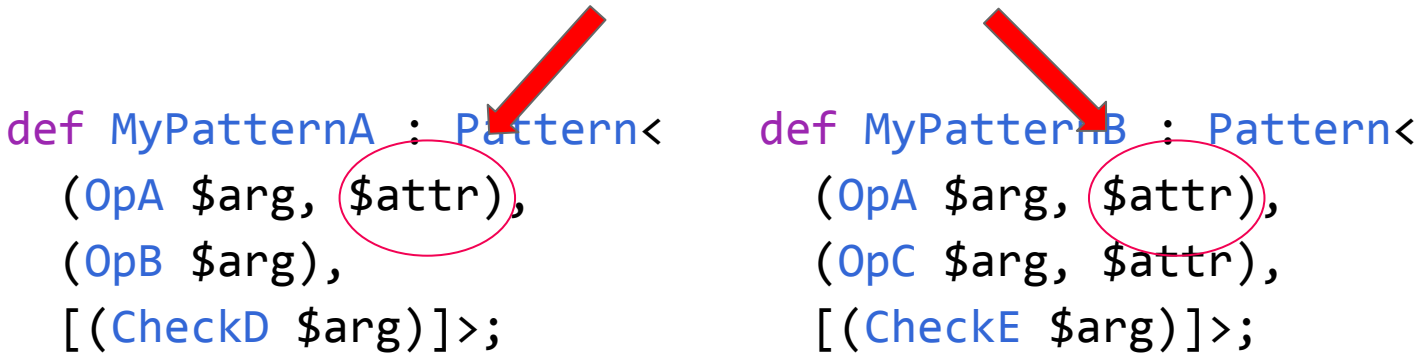
```
mlir-opt -opt-patterns=MyOptPatterns.pat
         -legalize-patterns=MyLegalizePatterns.pat
```

Duplicate Work

```
def MyPatternA : Pattern<
  (OpA $arg, $attr),
  (OpB $arg),
  [(CheckD $arg)]>;
```

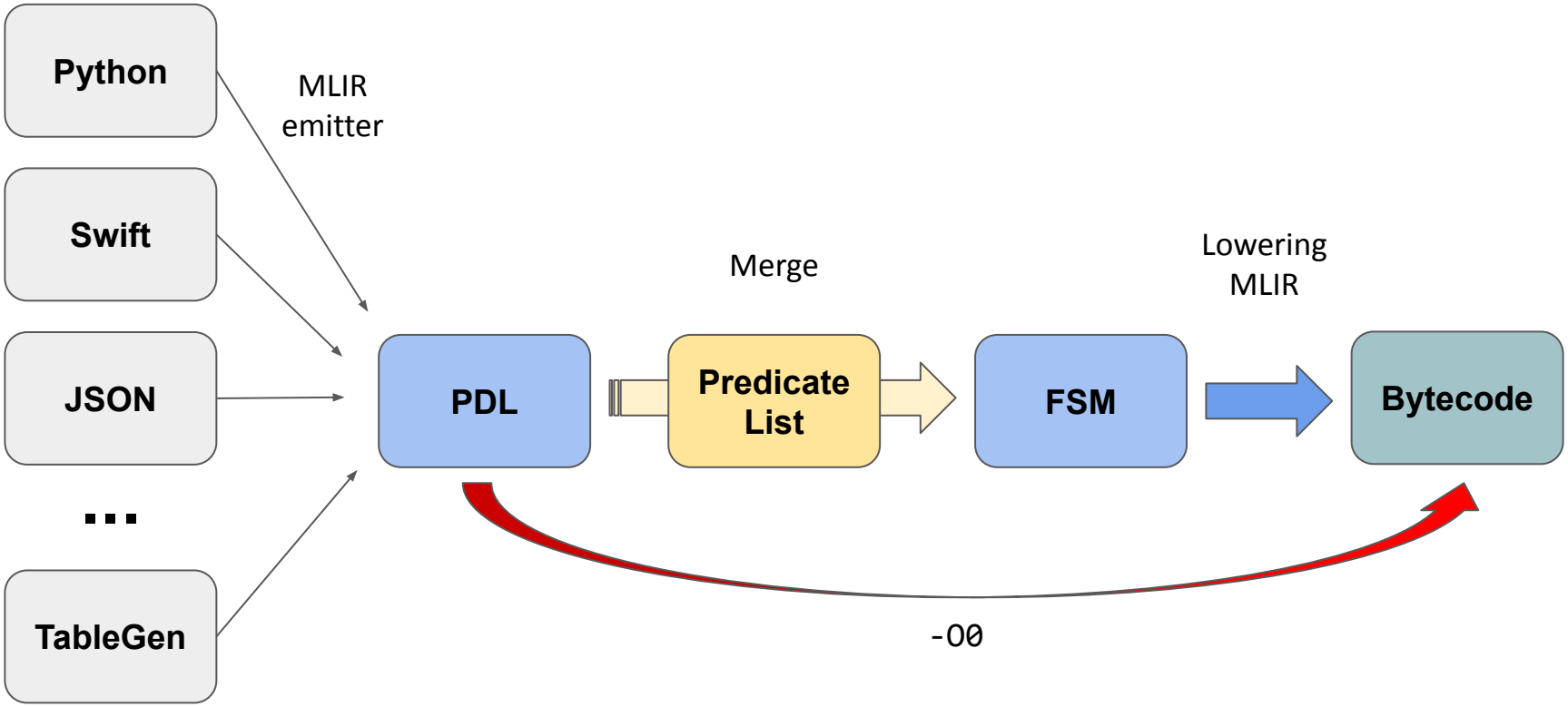
```
def MyPatternB : Pattern<
  (OpA $arg, $attr),
  (OpC $arg, $attr),
  [(CheckE $arg)]>;
```

Duplicate Work




```
op0.getAttr("myAttr")
```

Pattern Pipeline



Pattern Descriptor Language

Building MLIR with MLIR

```
def FusedMAdd : Pattern<(AddOp $x, (MulOp $y, $z)), (MAddOp $x, $y, $z)>;
def : Pattern<(MAddOp $x, $y, $z), (MAddIOp $x, $y, $z), [(isInt $x)]>;
def : Pattern<(MAddOp $x, $y, $z), (MAddFOp $x, $y, $z), [(isFlt $x)]>;
```

Building MLIR with MLIR

```
def FusedMAdd : Pattern<(AddOp $x, (MulOp $y, $z)), (MAddOp $x, $y, $z)>;
def : Pattern<(MAddOp $x, $y, $z), (MAddIOp $x, $y, $z), [(isInt $x)]>;
def : Pattern<(MAddOp $x, $y, $z), (MAddFOp $x, $y, $z), [(isFlt $x)]>;
```

```
pdl.pattern @FusedMAdd {
  %x, %y, %z = pdl.ins 3
  %mul_op, %mul_ret = pdl.node “u.mul”(%y, %z)
  %add_op, %add_ret = pdl.node “u.add”(%x, %mul_ret)
  pdl.output(%add_op) {
    %madd_op, %madd_ret = pdl.node “u.madd”(%x, %y, %z)
    pdl.replace -> (%madd_ret)
  }
}
```

```
pdl.pattern {
  %x, %y, %z = pdl.ins 3
  pdl.constraint “isFlt”(%x : value)
  %madd_op, %madd_ret = pdl.node “u.madd”(%x, %y, %z)
  pdl.output(%madd_op) {
    %maddf_op, %maddf_ret = pdl.node “u.maddf”(%x, %y, %z)
    pdl.replace -> (%maddf_ret)
  }
}

pdl.pattern {
  %x, %y, %z = pdl.ins 3
  pdl.constraint “isInt”(%x : value)
  %madd_op, %madd_ret = pdl.node “u.madd”(%x, %y, %z)
  pdl.output(%madd_op) {
    %maddi_op, %maddi_ret = pdl.node “u.maddi”(%x, %y, %z)
    pdl.replace -> (%maddi_ret)
  }
}
```

Building MLIR with MLIR

```
def FusedMAdd : Pattern<(AddOp $x, (MulOp $y, $z)), (MAddOp $x, $y, $z)>;
def : Pattern<(MAddOp $x, $y, $z), (MAddIOp $x, $y, $z), [(isInt $x)]>;
def : Pattern<(MAddOp $x, $y, $z), (MAddFOp $x, $y, $z), [(isFlt $x)]>;
```

```
pdl.pattern @FusedMAdd {
  %x, %y, %z = pdl.ins 3
  %mul_op, %mul_ret = pdl.node “u.mul”(%y, %z)
  %add_op, %add_ret = pdl.node “u.add”(%x, %mul_ret)
  pdl.output(%add_op) {
    %madd_op, %madd_ret = pdl.node “u.madd”(%x, %y, %z)
    pdl.replace -> (%madd_ret)
  }
}
```

```
pdl.pattern {
  %x, %y, %z = pdl.ins 3
  pdl.constraint “isFlt”(%x : value)
  %madd_op, %madd_ret = pdl.node “u.madd”(%x, %y, %z)
  pdl.output(%madd_op) {
    %maddf_op, %maddf_ret = pdl.node “u.maddf”(%x, %y, %z)
    pdl.replace -> (%maddf_ret)
  }
}

pdl.pattern {
  %x, %y, %z = pdl.ins 3
  pdl.constraint “isInt”(%x : value)
  %madd_op, %madd_ret = pdl.node “u.madd”(%x, %y, %z)
  pdl.output(%madd_op) {
    %maddi_op, %maddi_ret = pdl.node “u.maddi”(%x, %y, %z)
    pdl.replace -> (%maddi_ret)
  }
}
```

Building MLIR with MLIR

```
def FusedMAdd : Pattern<(AddOp $x, (MulOp $y, $z)), (MAddOp $x, $y, $z)>;
```

```
from mlir import pdl
from dialects import U
ins = pdl.Ins(3)
mul = pdl.Node(U.Mul, ins[1], ins[2])
add = pdl.Node(U.Add, ins[0], mul[0])
pdl.Replace(add, pdl.Node(U.MAdd, ins[0], ins[1], ins[2]))
```

```
// Verify patterns
mlir::verify(myPdlPatterns);
```

PDL Extensibility

```
%optional = pdl.attr ?
/* ... */
pdl.output(%root) {
  %defAttr = pdl.render attr “useDefault”<42 : i16>(%optional : attr)
  /* ... */
}
```

PDL Extensibility

```
%optional = pdl.attr ?
/* ... */
pdl.output(%root) {
  %defAttr = pdl.render attr “useDefault”<42 : i16>(%optional : attr)
  /* ... */
}
```

```
GenericValue useDefault(GenericValue arg,
                        Attribute defVal) {
  return arg ? arg : defVal;
}
```

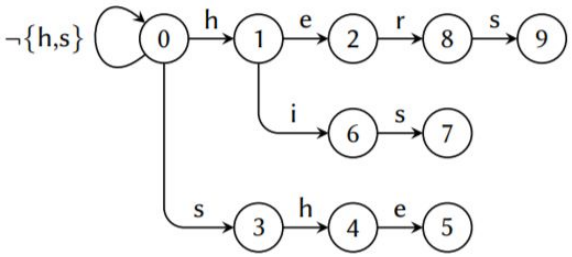
PDL Extensibility

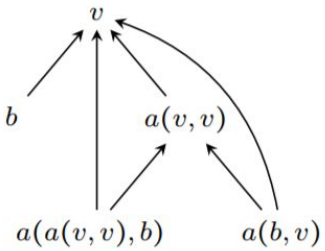
```
pdl.constraint “atLeastOneIsOneOf”<1, 2, 4, i16, “abc”, ...>(
  %attr0, %attr1, %attr2, ..., %attrN
)
```

```
bool atLeastOneIsOneOf(ArrayRef<GenericValue> args,
                       ArrayAttr params) {
  for (auto &arg : args)
    if (llvm::find(params, arg.get<Attribute>()) != params.end())
      return true;
  return false;
}
```

Pattern Optimization

The Literature





T_a 1 / 0	0	1	2	3	4
0	2	2	2	2	2
1	3	3	3	3	3
2	2	4	2	2	2
3	2	4	2	2	2
4	2	4	2	2	2

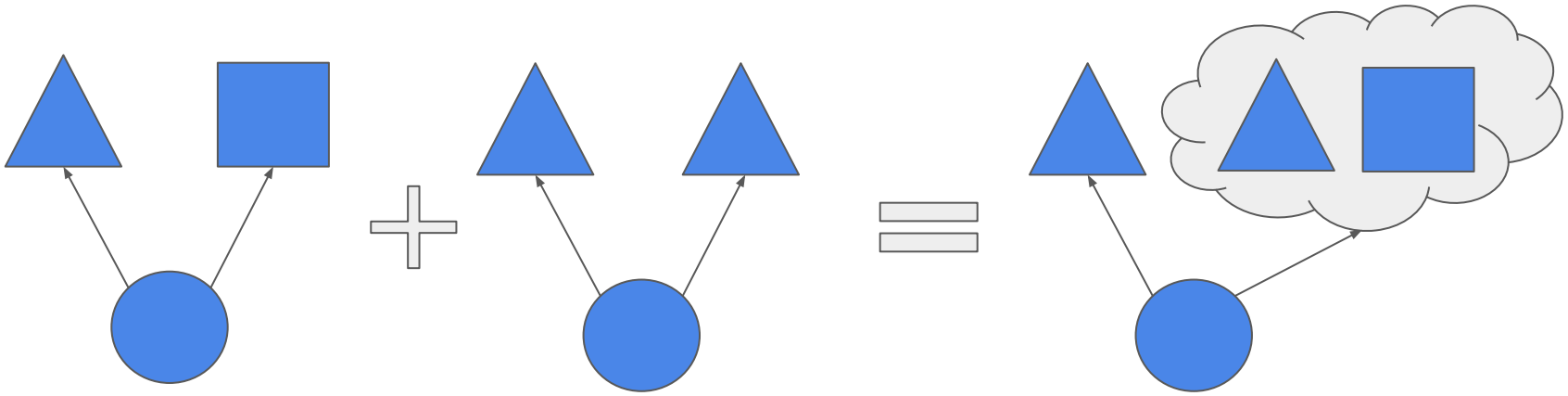
$\mu_{a,1}$	0	1	2	3	4
0	1	0	0	0	

$\mu_{a,0}$	
0	0
1	1
2	2
3	2
4	2

τ_a 1 / 0	0	1
0	2	2
1	3	3
2	2	4

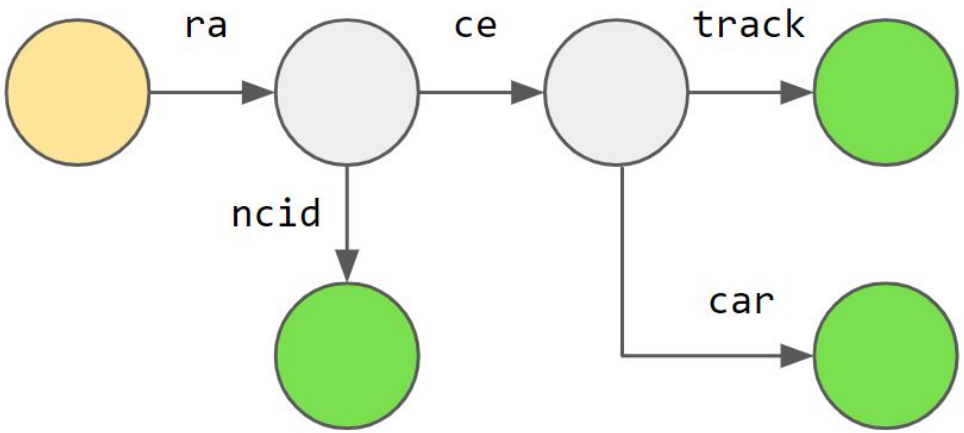
GENERATETABLE(set of subpatterns S, subsumption graph $\overline{G}_S$, symbol $a \in \Sigma$):
1: do topological sort on $\overline{G}_S$
2: initialize all entries in T_a with $v \in S$
3: **for each** $s = a(s_1, \ldots, s_m) \in S$ in increasing order of subsumption **do**
4: **for each** m-tuple $\langle s'_1, \ldots, s'_m \rangle$ s.t. for $1 \le i \le m$, $s'_i \ge s_i$ **do**
5: $T_a[s'_1, \ldots, s'm] = s$
6: **end for**
7: **end for**

Become as One



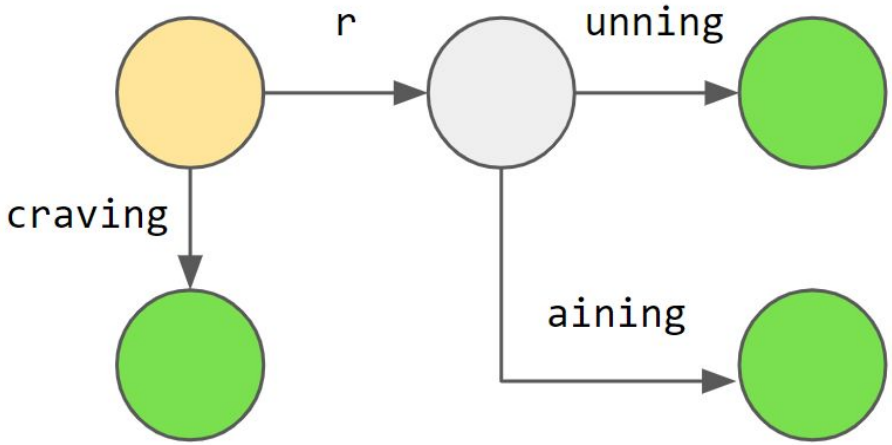
Strings are Patterns

```
["racecar",
 "racetrack",
 "rancid"]
```



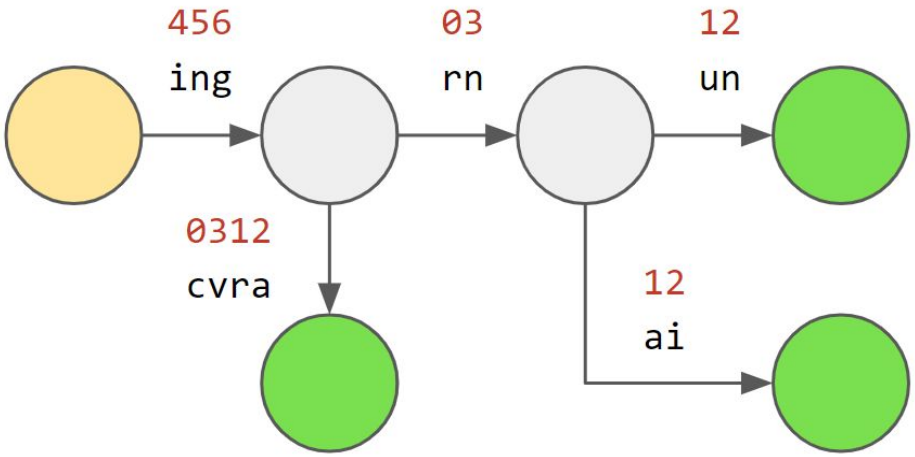
Strings are Patterns

```
[“running”,
 “raining”,
 “craving”]
```



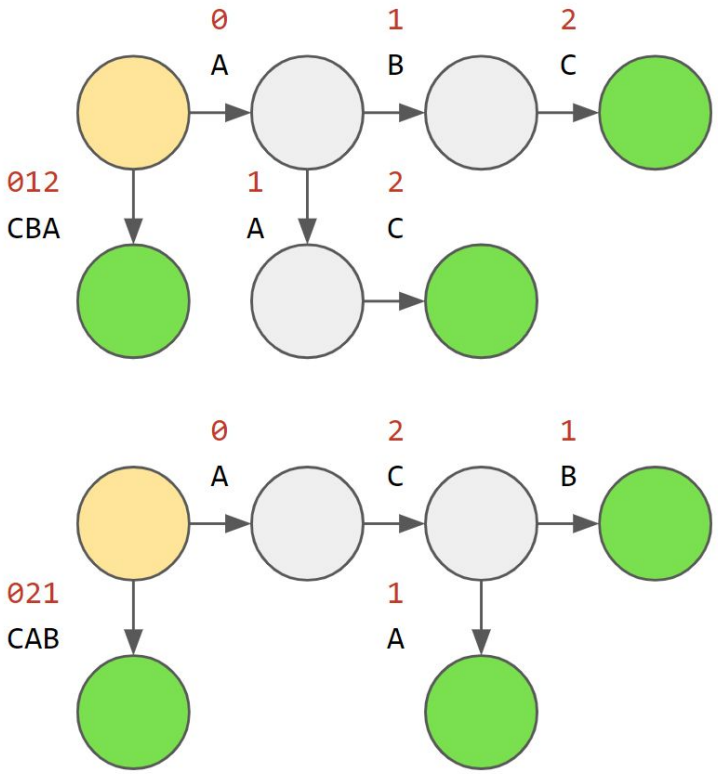
Strings are Patterns

```
[“running”,
 “raining”,
 “craving”]
```

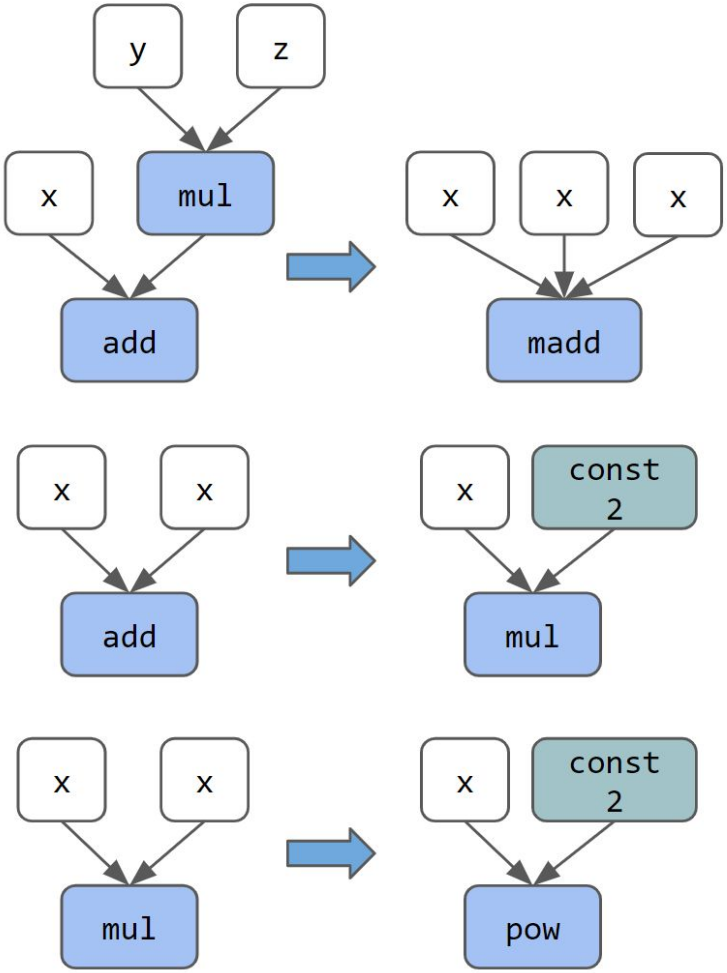


Strings are Patterns

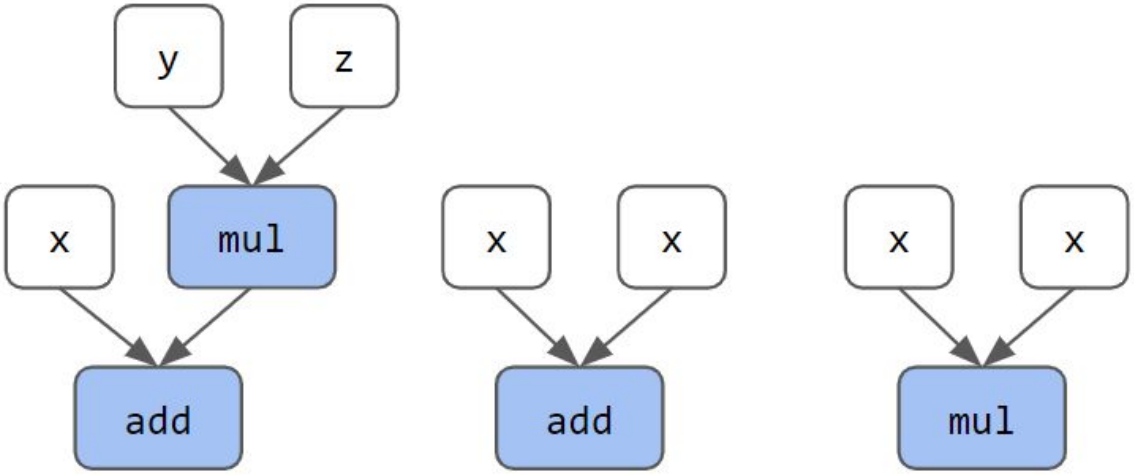
[“ABC”,
“AAC”,
“CBA”]



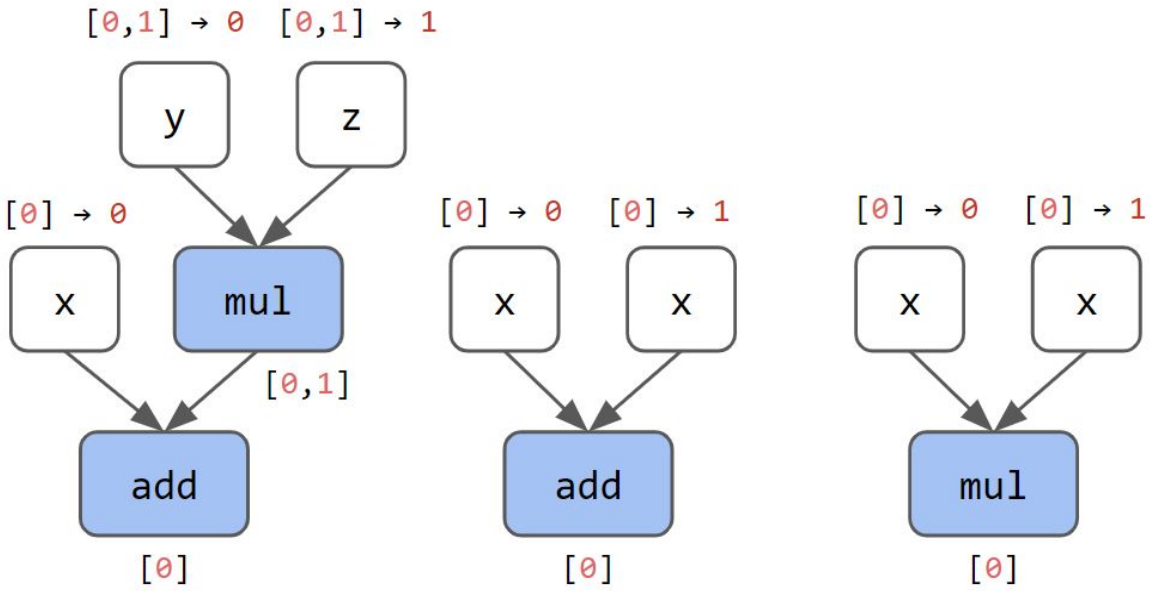
Patterns are Strings



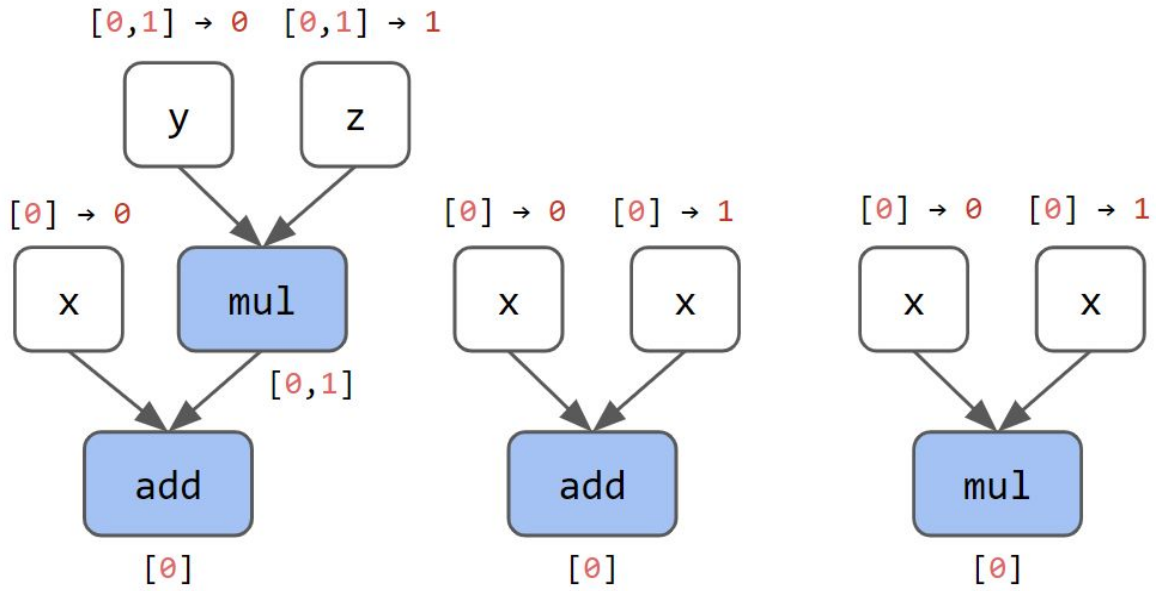
Patterns are Strings



Patterns are Strings



Patterns are Strings



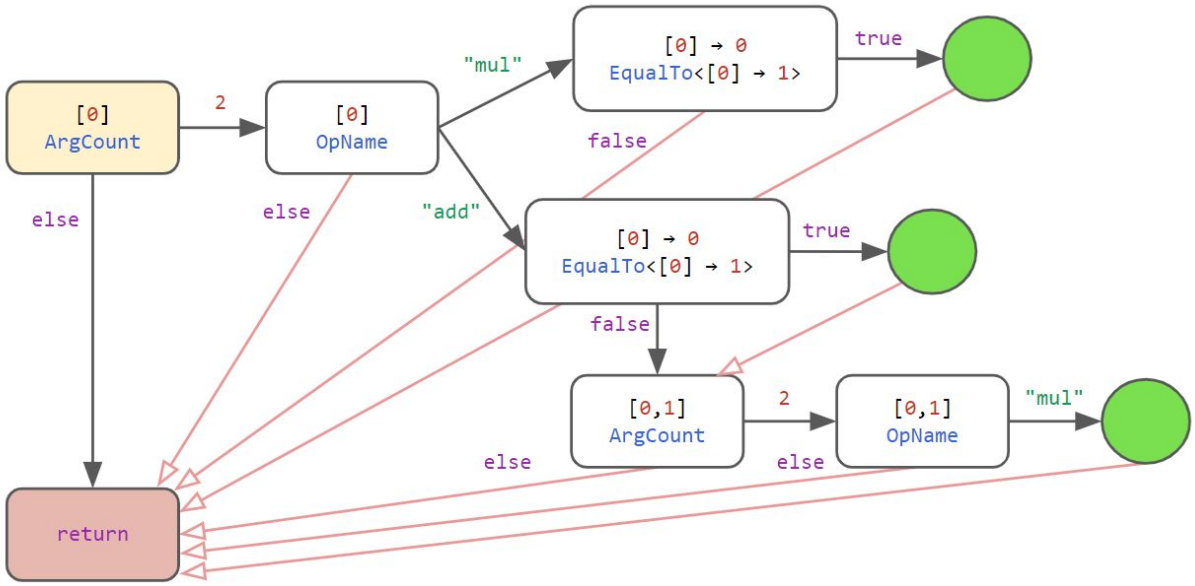

```
{[0], ArgCount, 2}
{[0], OpName, “add”}
{[0,1], ArgCount, 2}
{[0,1], OpName, “mul”}
```

```
{[0], ArgCount, 2}
{[0], OpName, “add”}
{[0] → 0,
EqualTo<[0] → 1>, true}
```

```
{[0], ArgCount, 2}
{[0], OpName, “mul”}
{[0] → 0,
EqualTo<[0] → 1>, true}
```

Pattern Matching FSM

```
{[0], ArgCount}
{[0], OpName}
{[0] → 0, EqualTo<[0] → 1>}
{[0,1], ArgCount}
{[0,1], OpName}
```

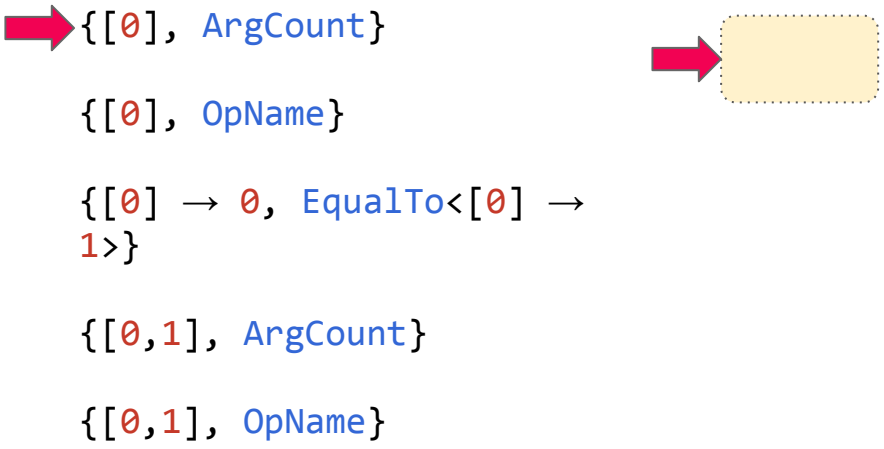


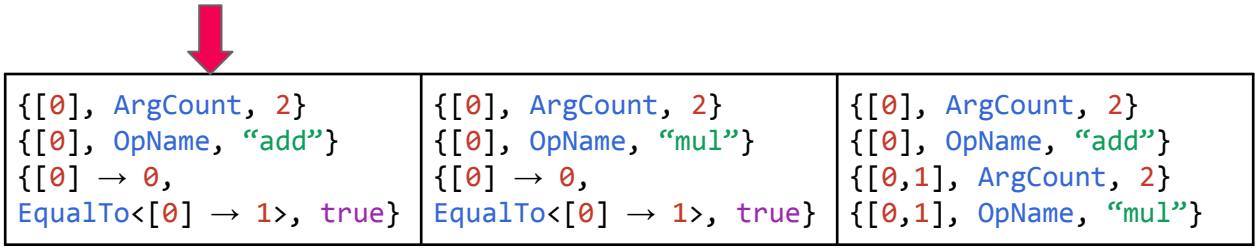
Pattern Matching FSM

```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```

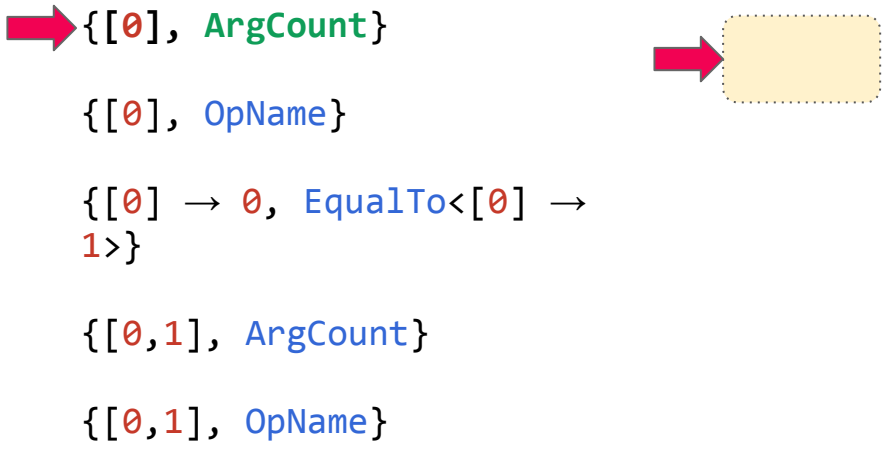
{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

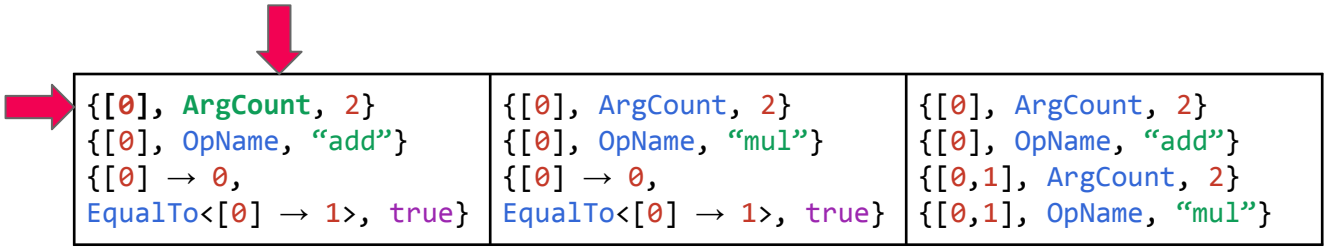
Pattern Matching FSM





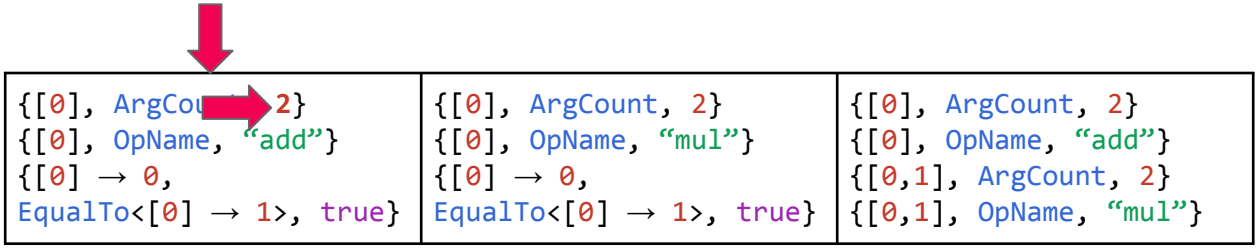
Pattern Matching FSM





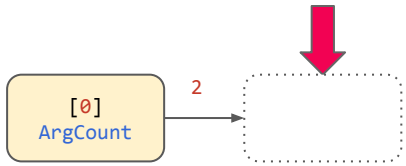
Pattern Matching FSM

```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



Pattern Matching FSM

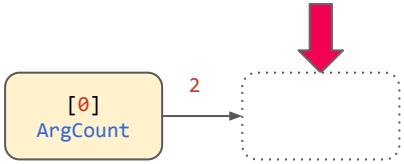
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

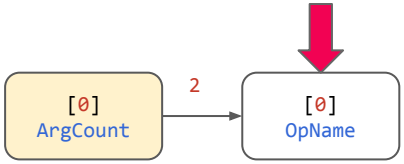
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

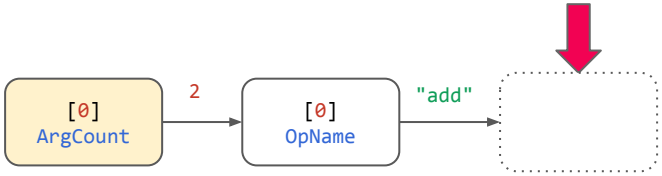
{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

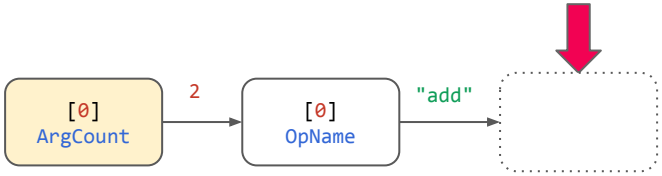
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

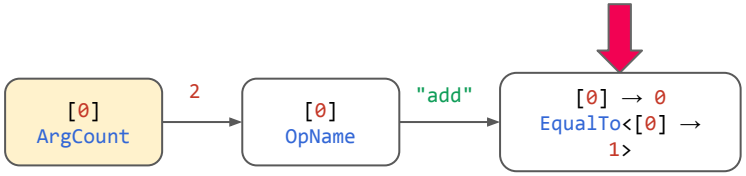
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 0>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

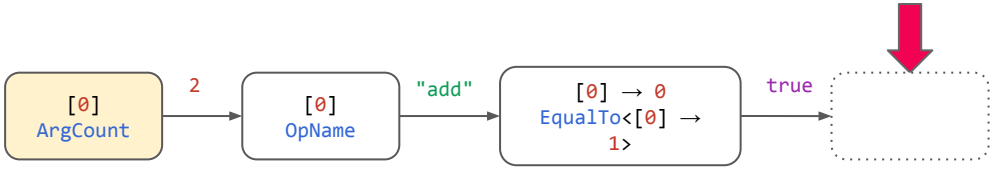
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

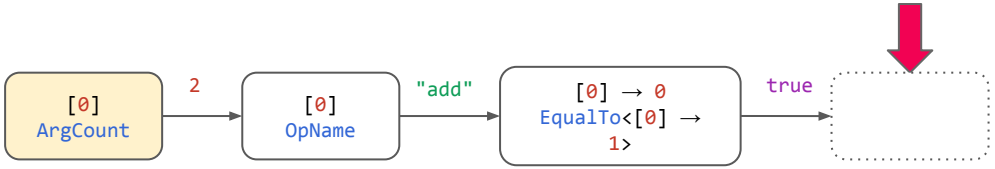
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

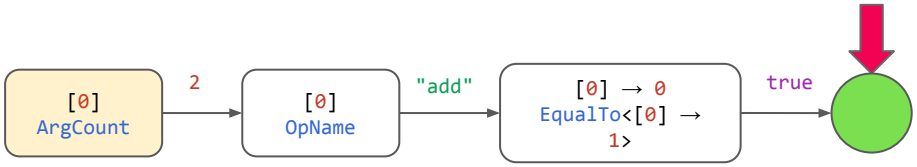
{[0,1], OpName}

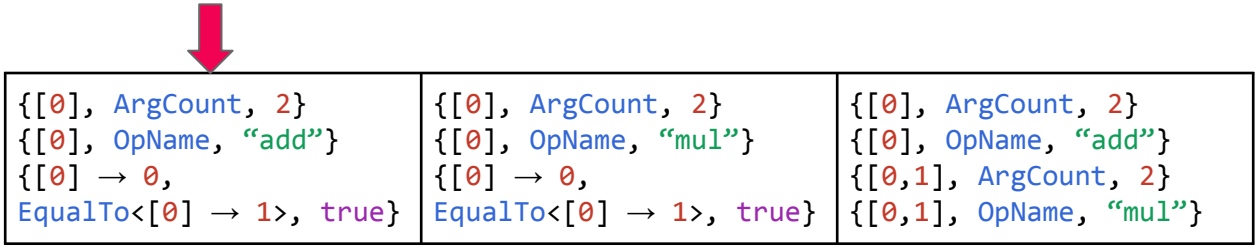


{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

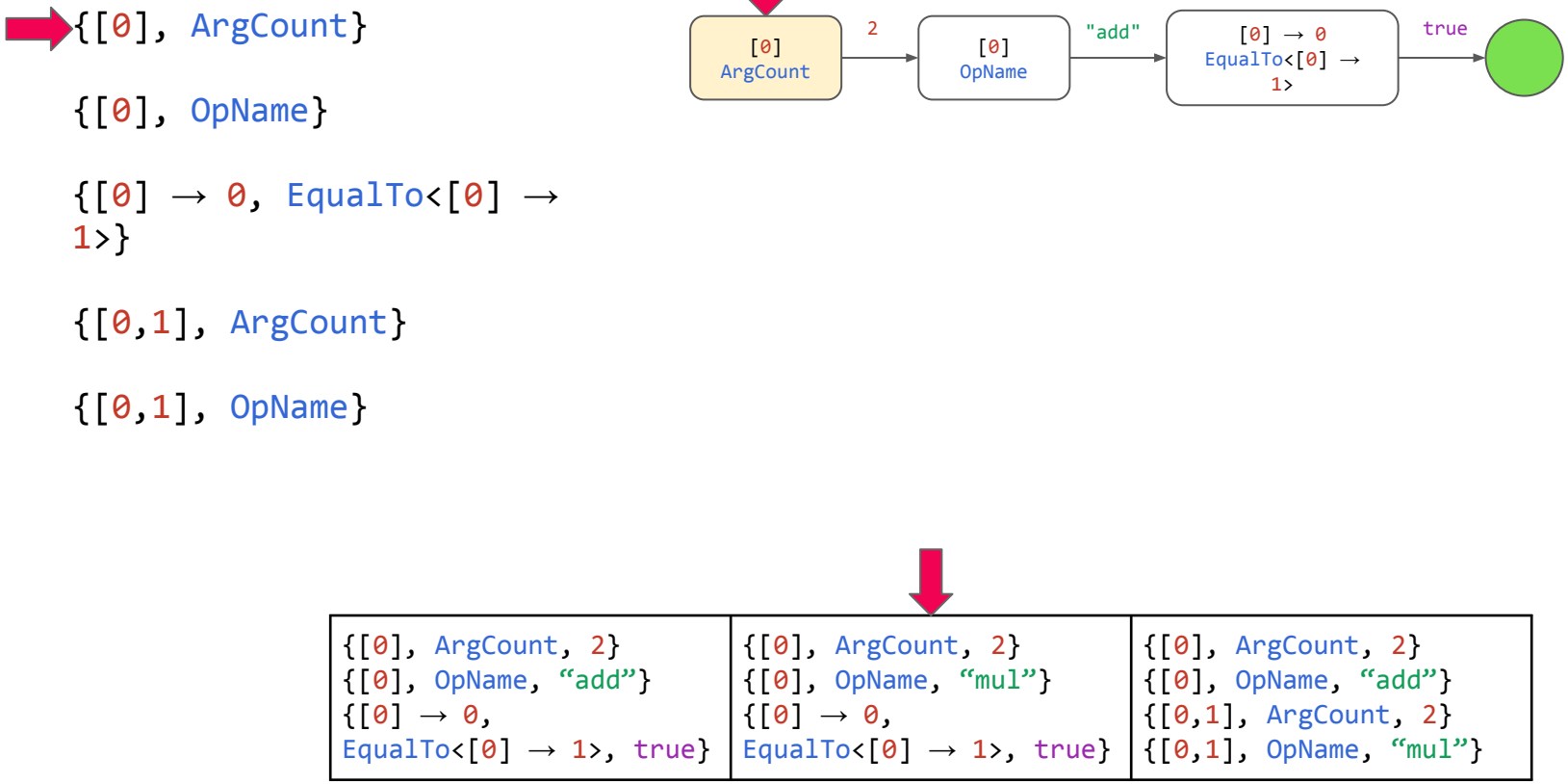
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```





{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM



Pattern Matching FSM

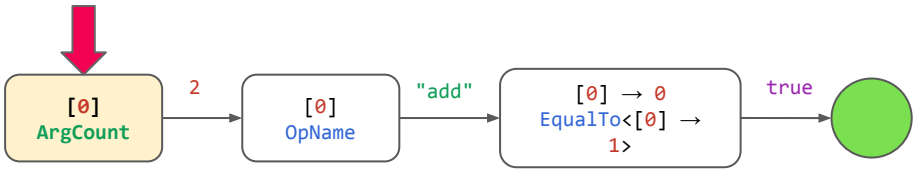
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

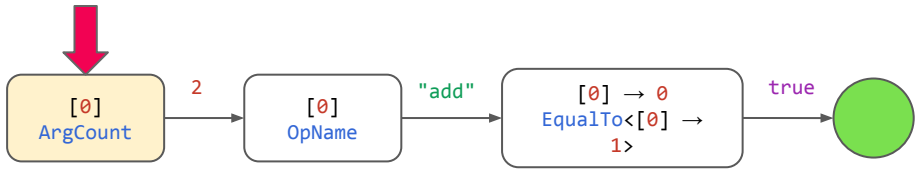
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

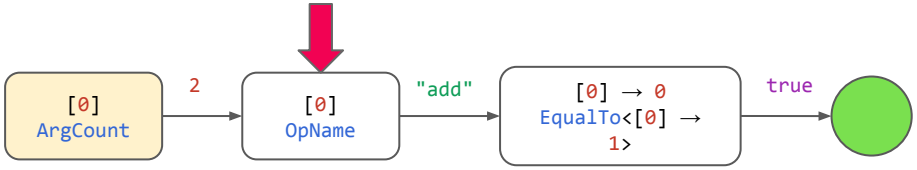
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

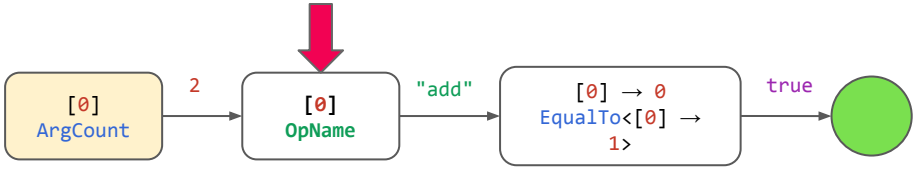
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

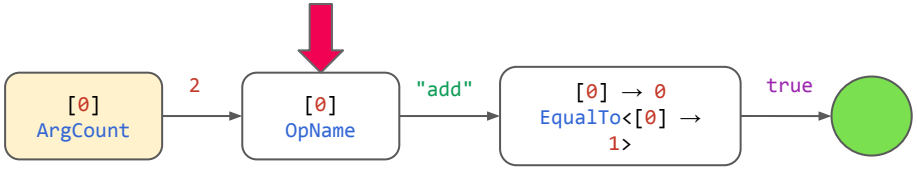
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

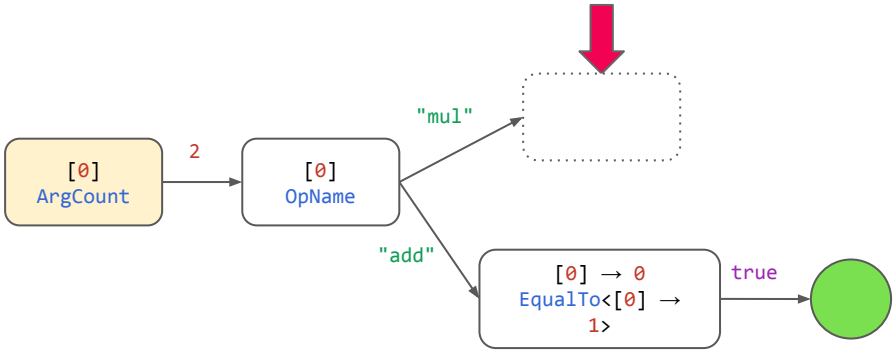
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

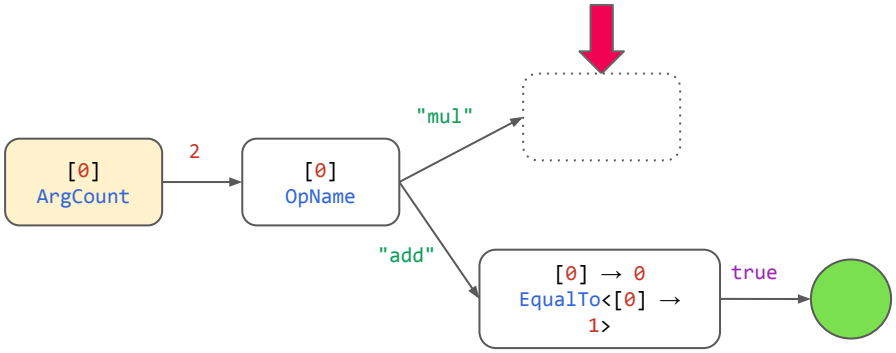
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

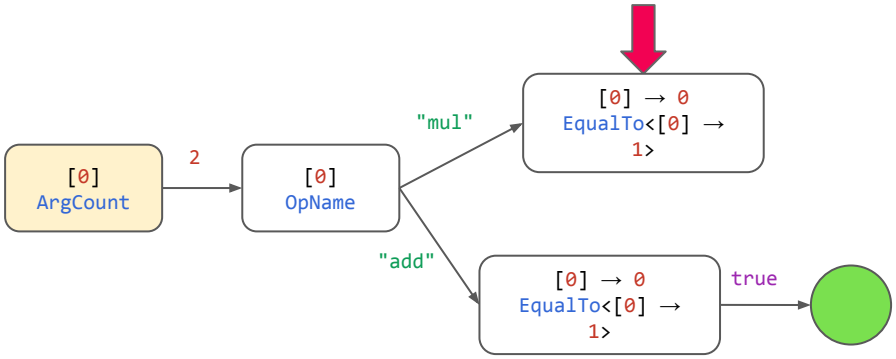
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

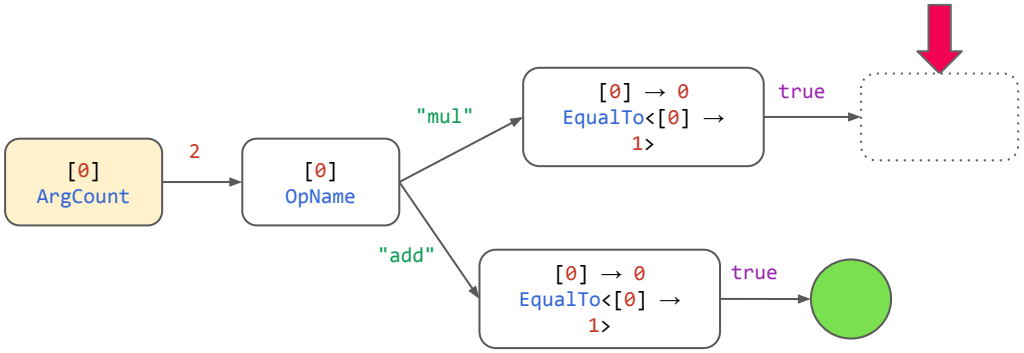
{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

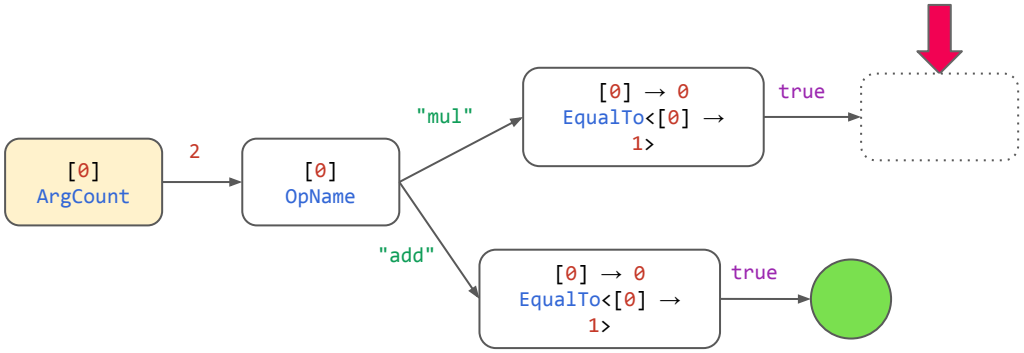
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

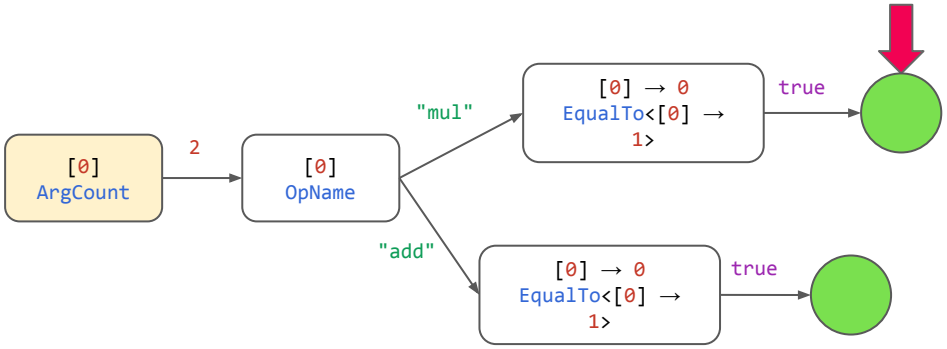
{[0,1], OpName}

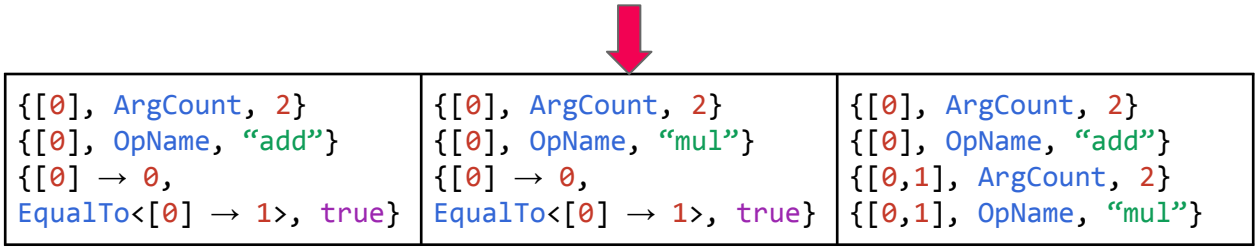


{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

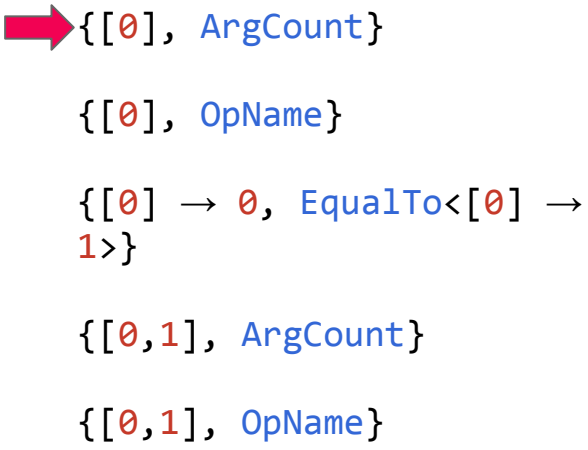
Pattern Matching FSM

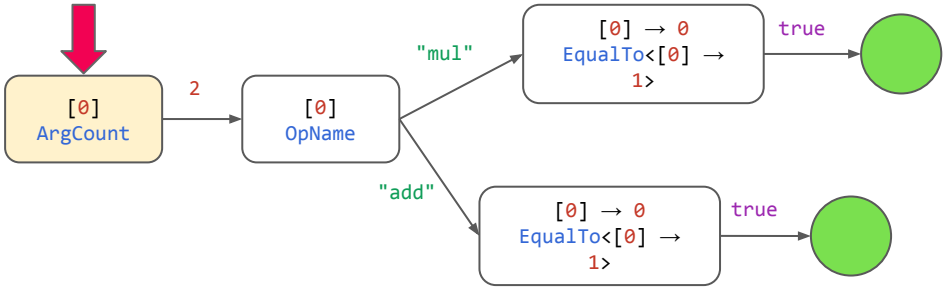
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```

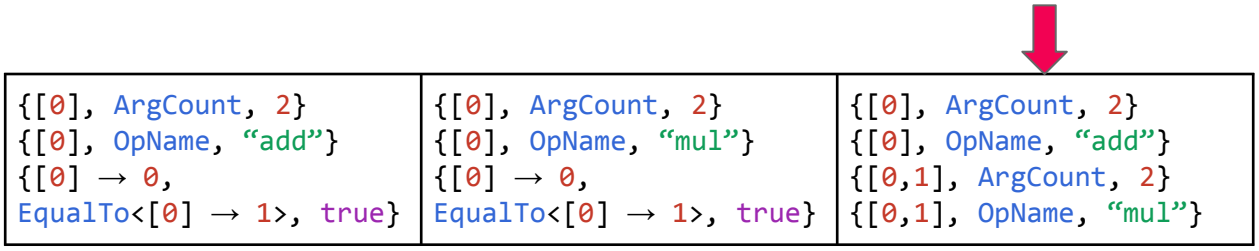




Pattern Matching FSM







Pattern Matching FSM

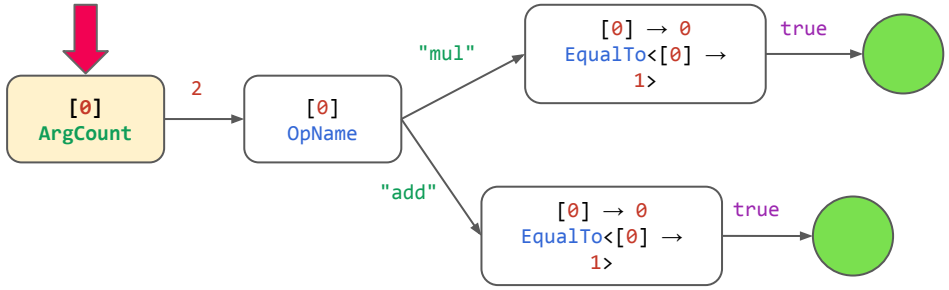
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

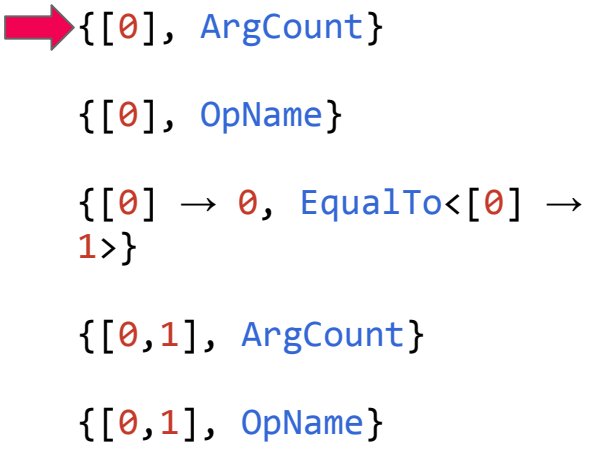
{[0,1], ArgCount}

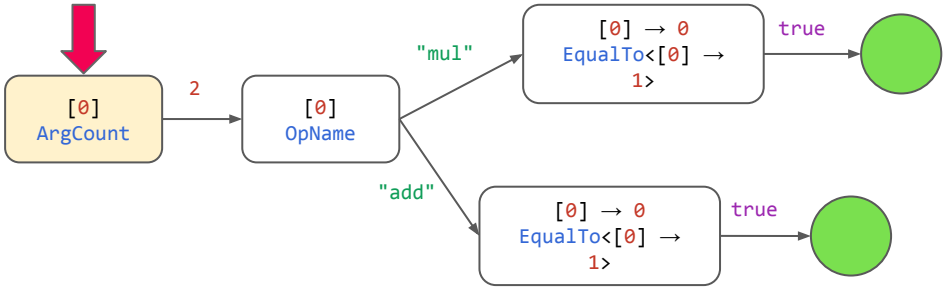
{[0,1], OpName}

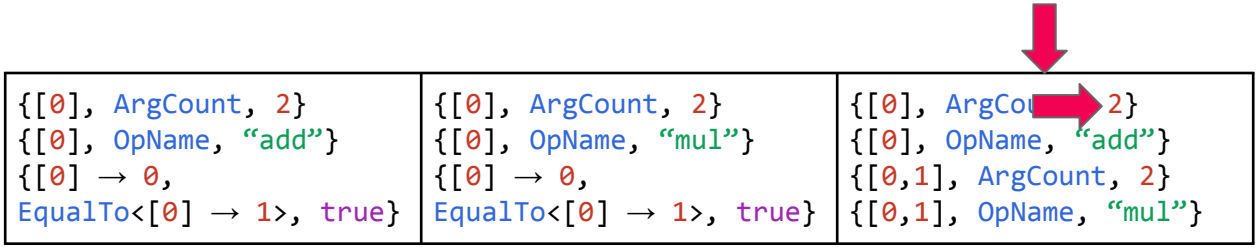


{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM







Pattern Matching FSM

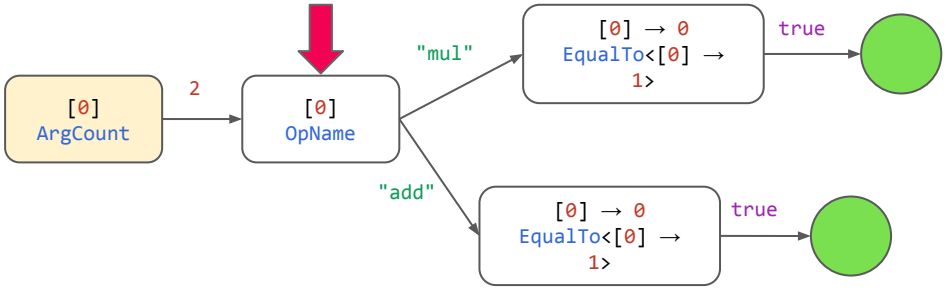
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

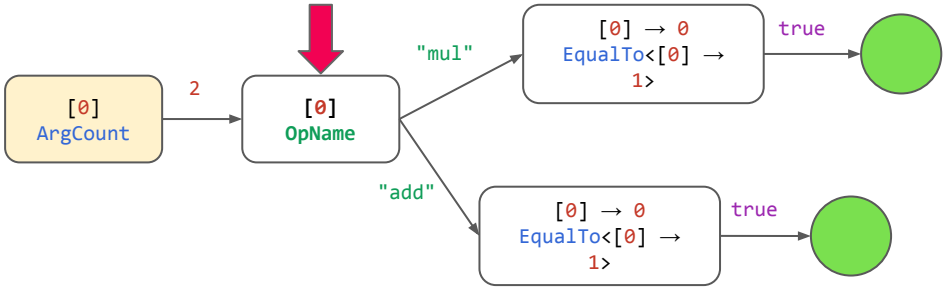
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

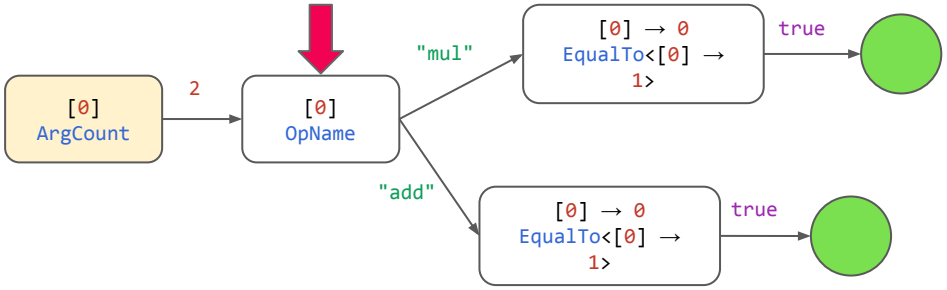
{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

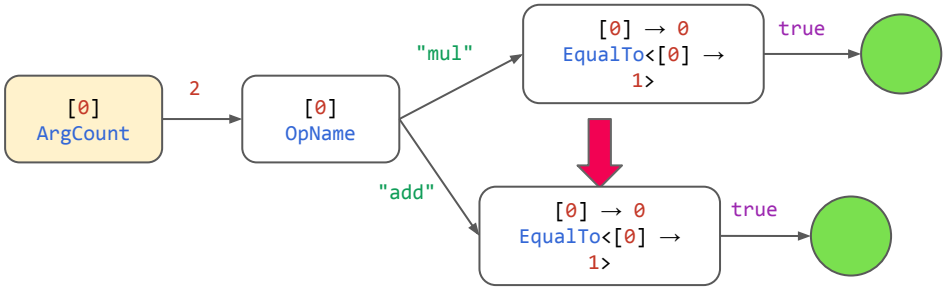
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

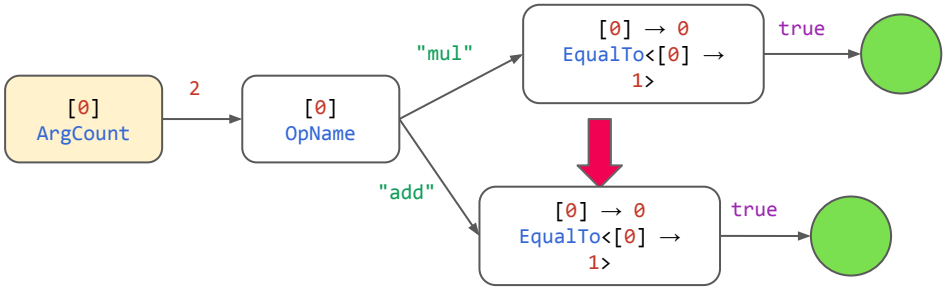
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

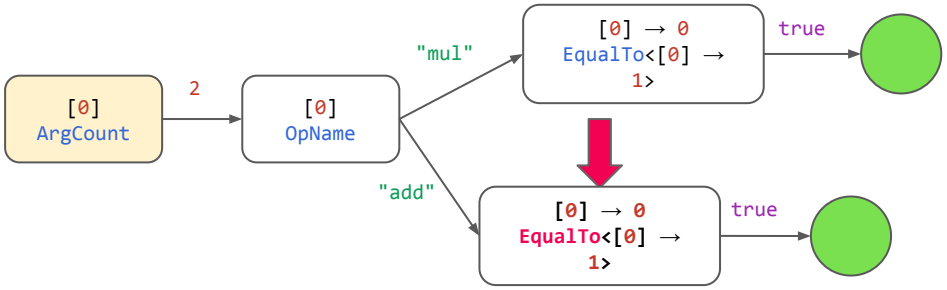
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

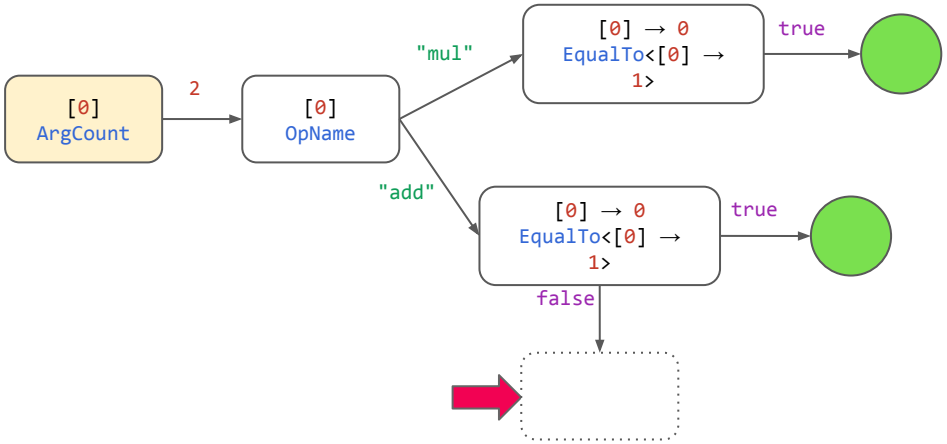
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

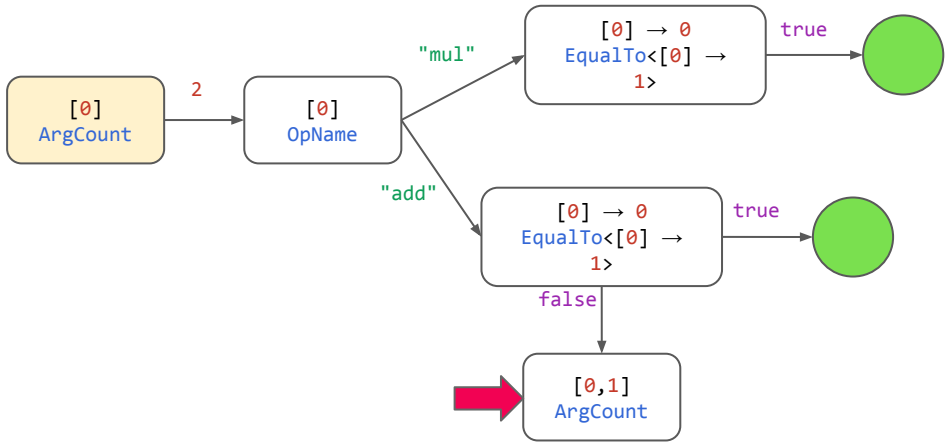
{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

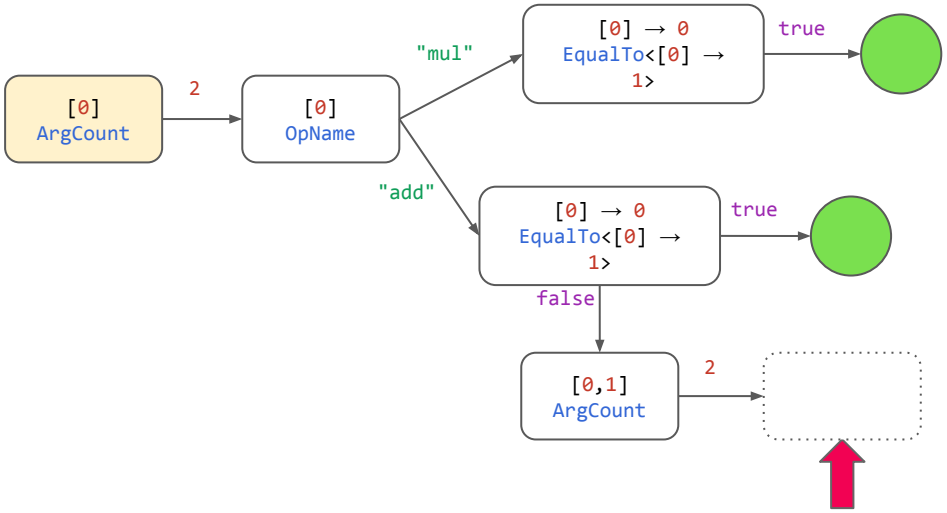
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

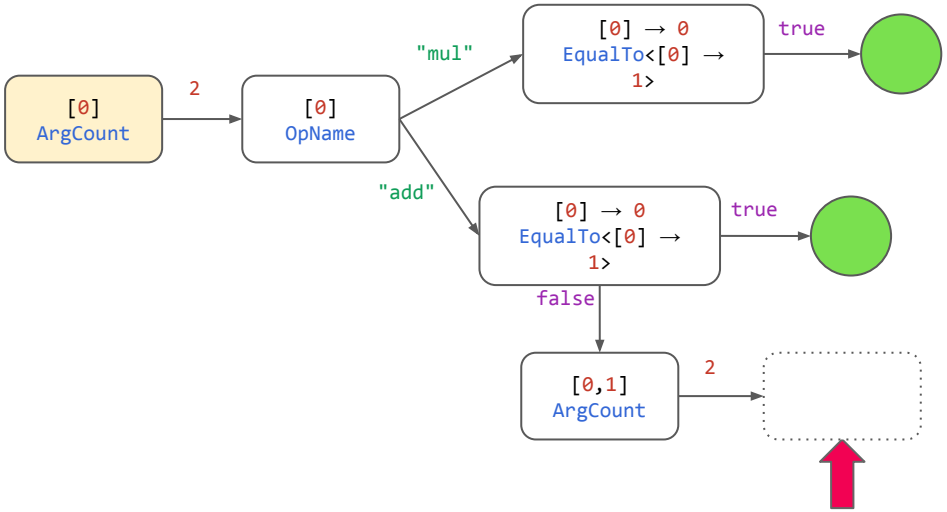
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

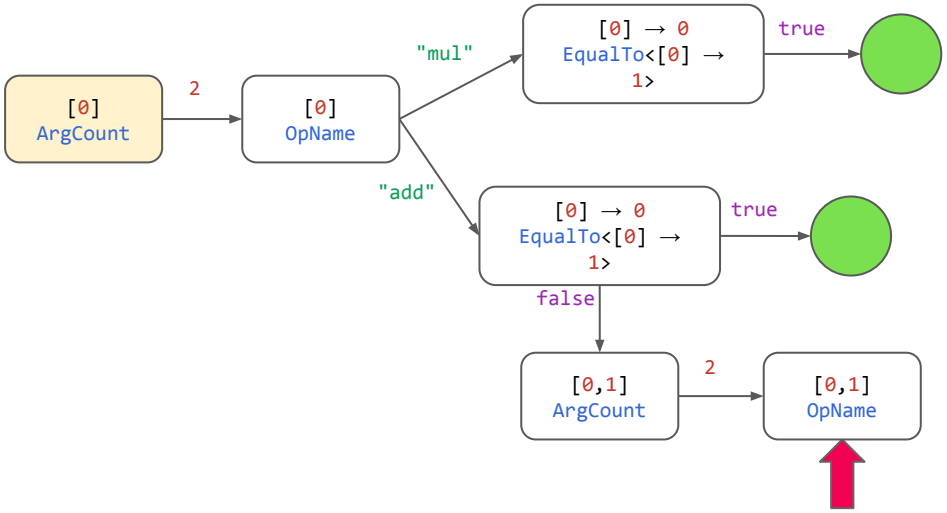
{[0,1], OpName}



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, tr	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

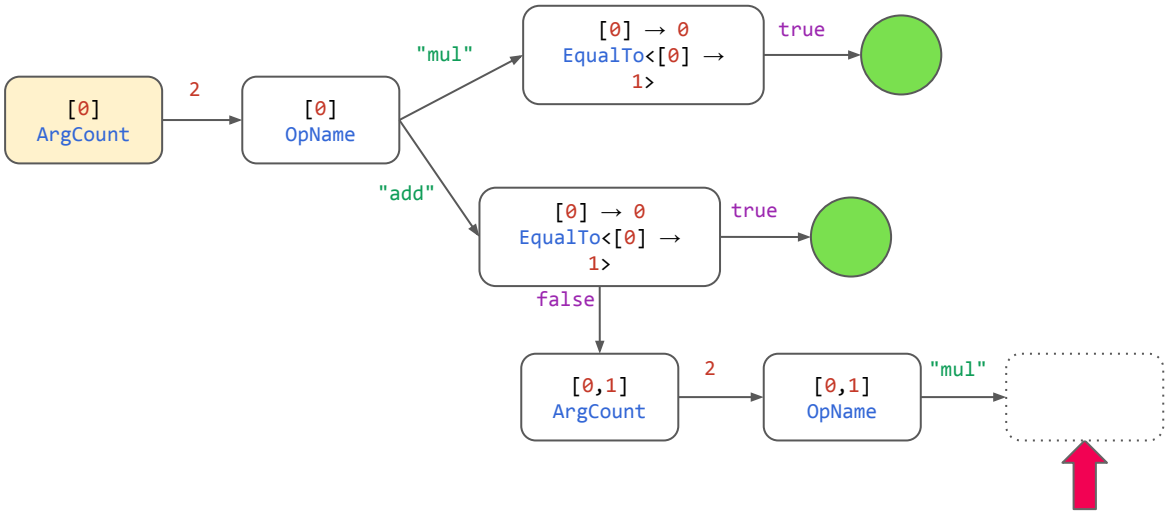
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

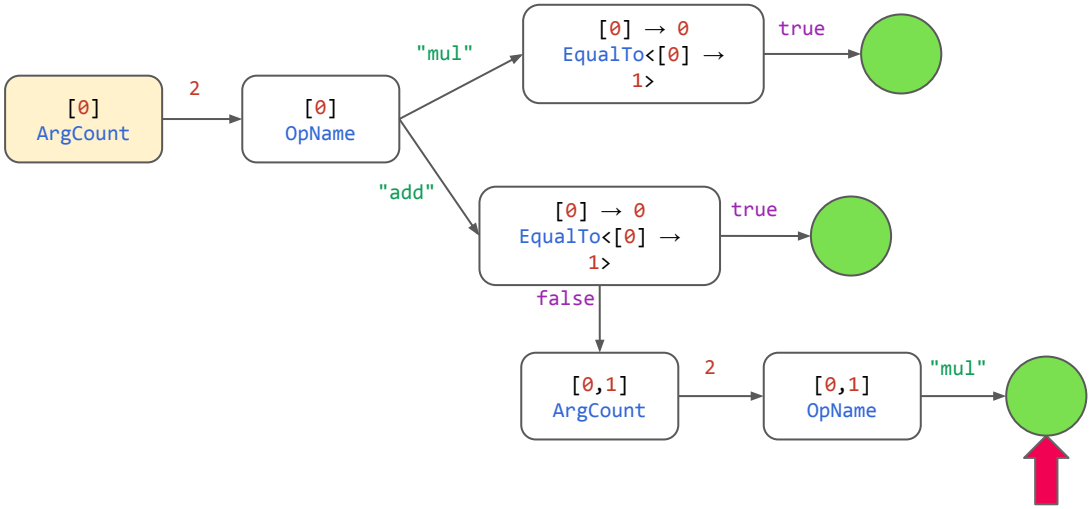
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

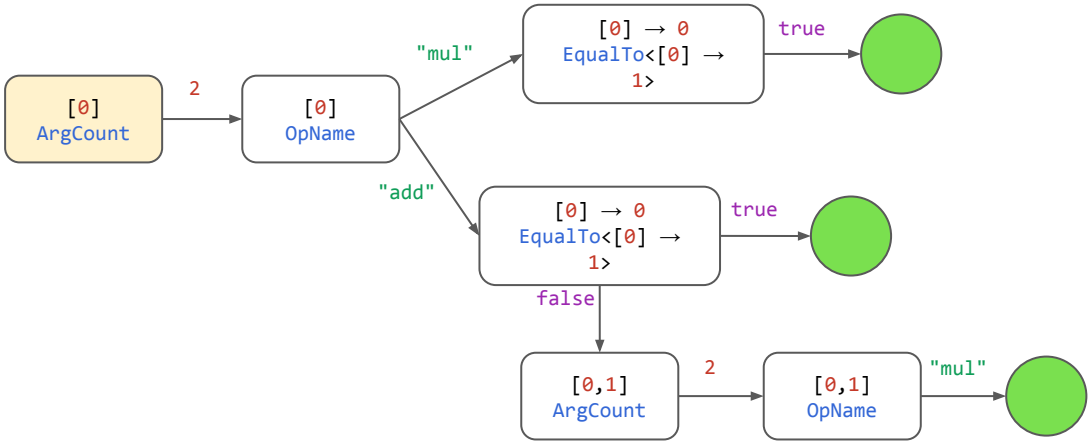
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

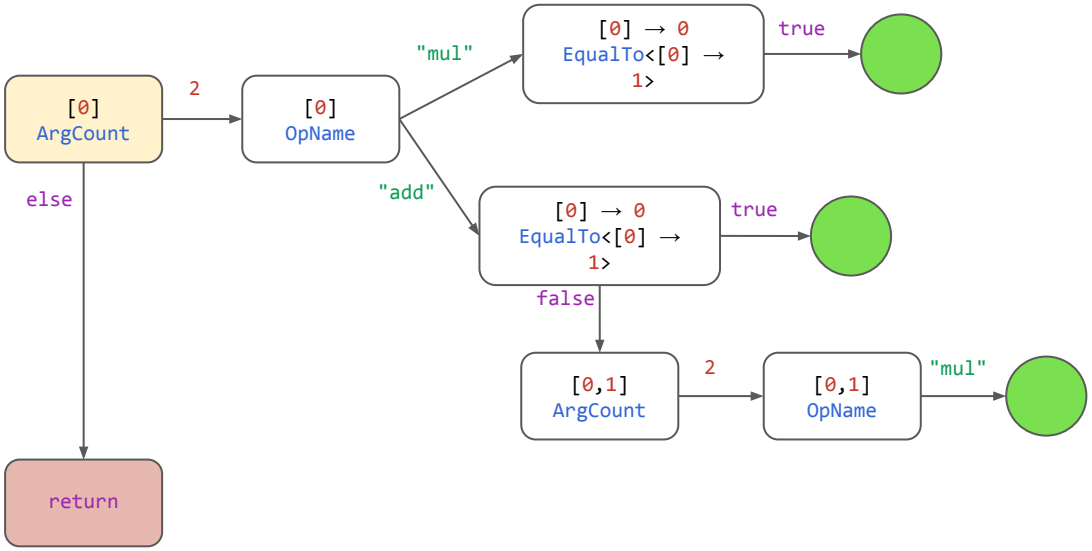
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] →
1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

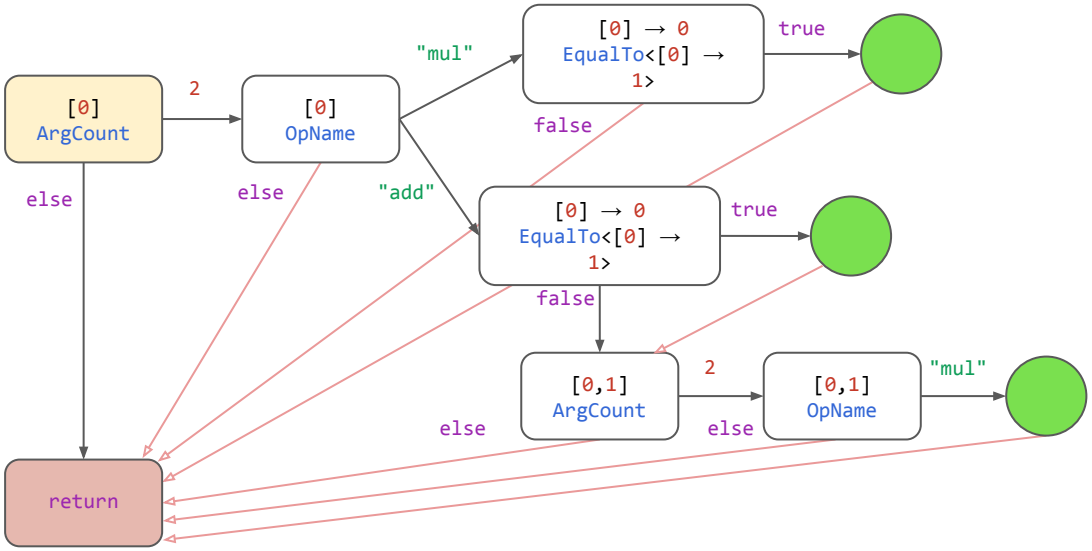
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Pattern Matching FSM

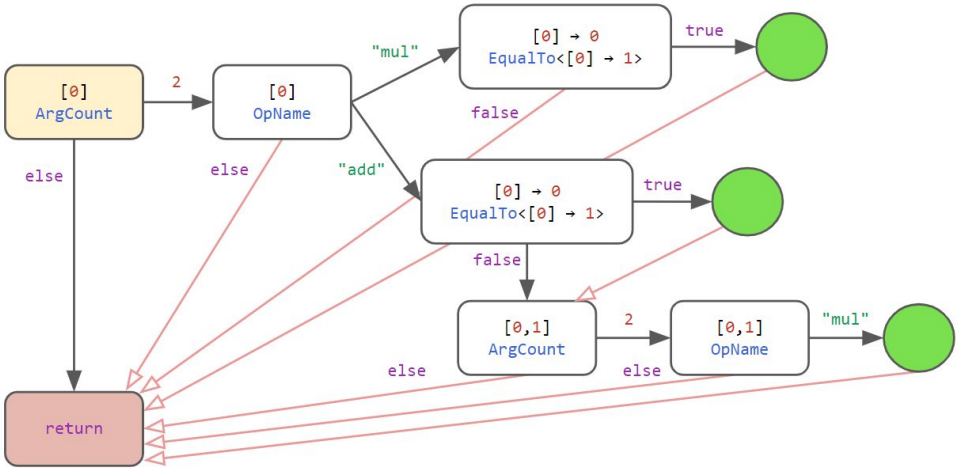
```
{[0], ArgCount}

{[0], OpName}

{[0] → 0, EqualTo<[0] → 1>}

{[0,1], ArgCount}

{[0,1], OpName}
```



{[0], ArgCount, 2} {[0], OpName, "add"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "mul"} {[0] → 0, EqualTo<[0] → 1>, true}	{[0], ArgCount, 2} {[0], OpName, "add"} {[0,1], ArgCount, 2} {[0,1], OpName, "mul"}

Interpreter Dialect

CFG → MLIR




```
func @matcher(%0 : !Operation) {
^bb0:
  CheckArgCount(%0) [^ex0, ^bb1] {count = 2}
^bb1:
  SwitchOpName(%0) [^ex0, ^bb2, ^bb3]
      {names = ["mul", "add"]}
^bb2:
  %1 = GetOperand(%0) {index = 0}
  %2 = GetOperand(%0) {index = 1}
  EqualTo(%1, %2) [^ex0, ^rr3]
^rr0:
  RegisterResult() [^ex0] {id = 0}
^bb3:
  %3 = GetOperand(%0) {index = 0}
  %4 = GetOperand(%0) {index = 1}
  EqualTo(%3, %4) [^bb4, ^rr1]
^rr1:
  RegisterResult() [^bb4] {id = 1}
^bb4:
  %5 = GetDefiningOp(%4)
  CheckArgCount(%5) [^ex0, ^bb5] {count = 2}
^bb5:
  CheckOpName(%5) [^ex0, ^rr2] {name = "mul"}
^rr2:
  RegisterResult() [^ex0] {id = 2}
^ex0:
  return
}
```

Lowering to Byte-Code

```
func @matcher(%0 : !Operation) {
^bb0:
  CheckArgCount(%0) [^ex0, ^bb1] {count = 2}
^bb1:
  SwitchOpName(%0) [^ex0, ^bb2, ^bb3]
      {names = ["mul", "add"]}
^bb2:
  %1 = GetOperand(%0) {index = 0}
  %2 = GetOperand(%0) {index = 1}
  EqualTo(%1, %2) [^ex0, ^rr0]
^rr0:
  RegisterResult() [^ex0] {id = 0}
^bb3:
  %3 = GetOperand(%0) {index = 0}
  %4 = GetOperand(%0) {index = 1}
  EqualTo(%3, %4) [^bb4, ^rr1]
^rr1:
  RegisterResult() [^bb4] {id = 1}
^bb4:
  %5 = GetDefiningOp(%4)
  CheckArgCount(%5) [^ex0, ^bb5] {count = 2}
^bb5:
  CheckOpName(%5) [^ex0, ^rr2] {name = "mul"}
^rr2:
  RegisterResult() [^ex0] {id = 2}
^ex0:
  return
}
```

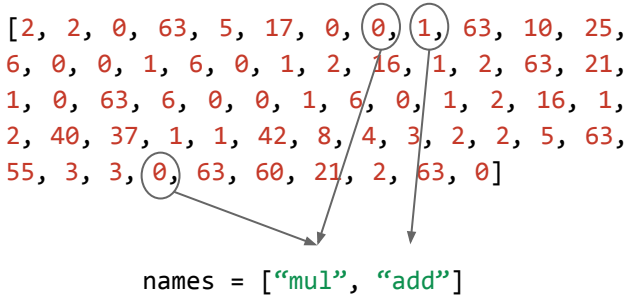
```
FuncOp matcher = …
matcher.walk(convertToByteCode);
```

```
[2, 2, 0, 63, 5, 17, 0, 0, 1, 63, 10, 25,
6, 0, 0, 1, 6, 0, 1, 2, 16, 1, 2, 63, 21,
1, 0, 63, 6, 0, 0, 1, 6, 0, 1, 2, 16, 1,
2, 40, 37, 1, 1, 42, 8, 4, 3, 2, 2, 5, 63,
55, 3, 3, 0, 63, 60, 21, 2, 63, 0]
```

```
names = [“mul”, “add”]
```

Static Info

```
func @matcher(%0 : !Operation) {
^bb0:
  CheckArgCount(%0) [^ex0, ^bb1] {count = 2}
^bb1:
  SwitchOpName(%0) [^ex0, ^bb2, ^bb3]
      {names = ["mul", "add"]}
^bb2:
  %1 = GetOperand(%0) {index = 0}
  %2 = GetOperand(%0) {index = 1}
  EqualTo(%1, %2) [^ex0, ^rr0]
^rr0:
  RegisterResult() [^ex0] {id = 0}
^bb3:
  %3 = GetOperand(%0) {index = 0}
  %4 = GetOperand(%0) {index = 1}
  EqualTo(%3, %4) [^bb4, ^rr1]
^rr1:
  RegisterResult() [^bb4] {id = 1}
^bb4:
  %5 = GetDefiningOp(%4)
  CheckArgCount(%5) [^ex0, ^bb5] {count = 2}
^bb5:
  CheckOpName(%5) [^ex0, ^rr2] {name = "mul"}
^rr2:
  RegisterResult() [^ex0] {id = 2}
^ex0:
  return
}
```



Memory Allocation

```
func @matcher(%0 : !Operation) {
^bb0:
  CheckArgCount(%0) [^ex0, ^bb1] {count = 2}
^bb1:
  SwitchOpName(%0) [^ex0, ^bb2, ^bb3]
      {names = ["mul", "add"]}
^bb2:
  %1 = GetOperand(%0) {index = 0}
  %2 = GetOperand(%0) {index = 1}
  EqualTo(%1, %2) [^ex0, ^rr0]
^rr0:
  RegisterResult() [^ex0] {id = 0}
^bb3:
  %3 = GetOperand(%0) {index = 0}
  %4 = GetOperand(%0) {index = 1}
  EqualTo(%3, %4) [^bb4, ^rr1]
^rr1:
  RegisterResult() [^bb4] {id = 1}
^bb4:
  %5 = GetDefiningOp(%4)
  CheckArgCount(%5) [^ex0, ^bb5] {count = 2}
^bb5:
  CheckOpName(%5) [^ex0, ^rr2] {name = "mul"}
^rr2:
  RegisterResult() [^ex0] {id = 2}
^ex0:
  return
}
```

```
[2, 2, 0, 63, 5, 17, 0, 0, 1, 63, 10, 25,
6, 0, 0, 1, 6, 0, 1, 2, 16, 1, 2, 63, 21,
1, 0, 63, 6, 0, 0, 1, 6, 0, 1, 2, 16, 1,
2, 40, 37, 1, 1, 42, 8, 4, 3, 2, 2, 5, 63,
55, 3, 3, 0, 63, 60, 21, 2, 63, 0]
```

```
names = [“mul”, “add”]
```

Comparison to LLVM

`SelectionDagISel`

- Contract nodes
- Factor nodes
- `vector<vector<...>>`

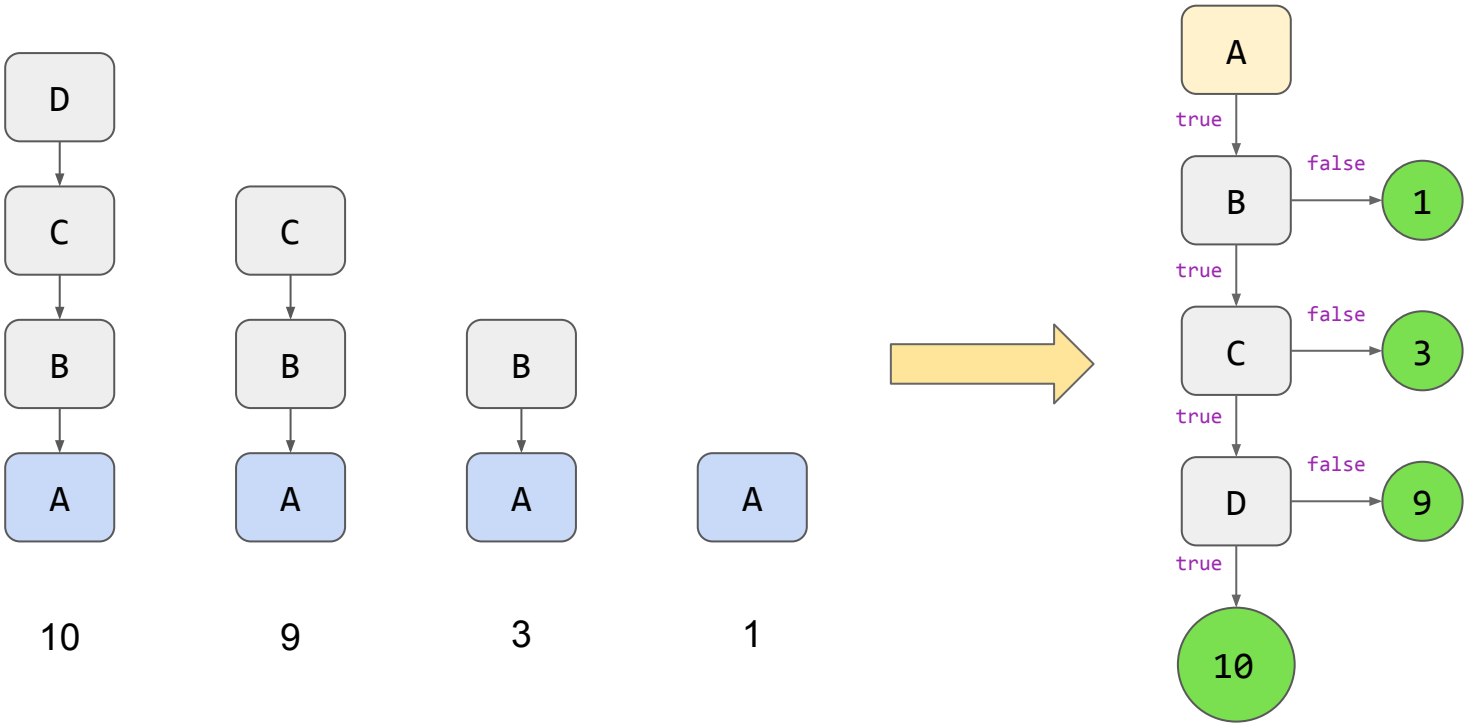
`GlobalISel`

- Direct improvement over **`SelectionDagISel`**
- Plus some small optimizations
- Uses a single `vector<...>`

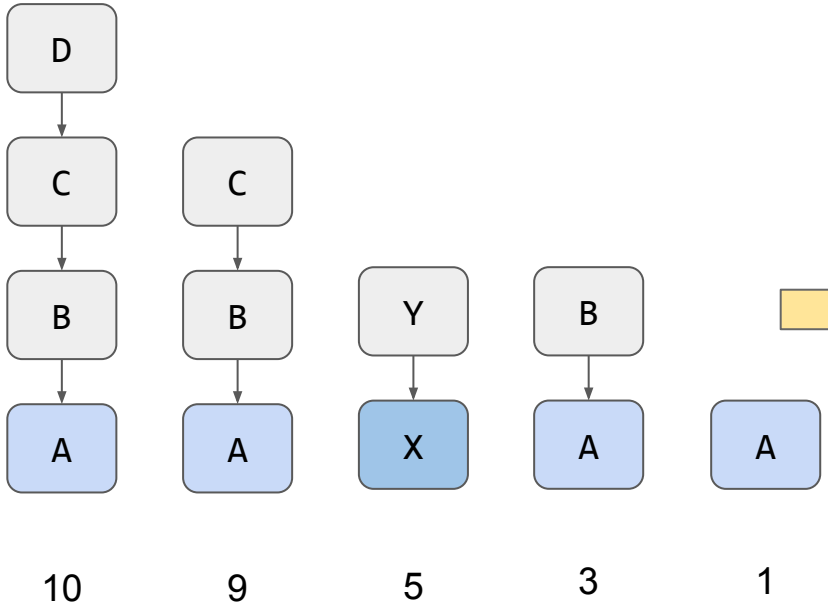
MLIR

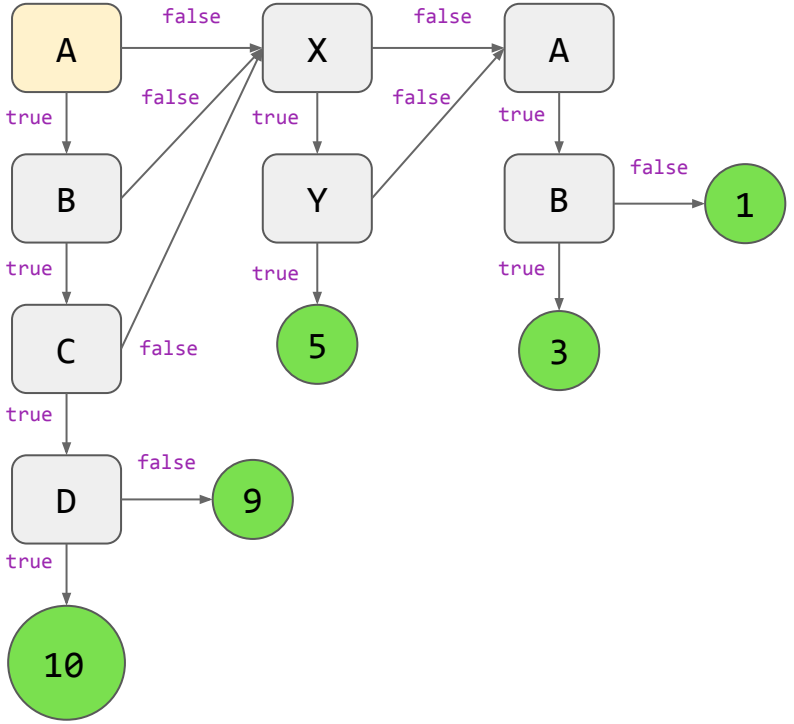
- More optimal merge/factor of nodes
- A few rewrite patterns on FSM
- Preallocated `vector<...>`

Merge Barriers



Merge Barriers



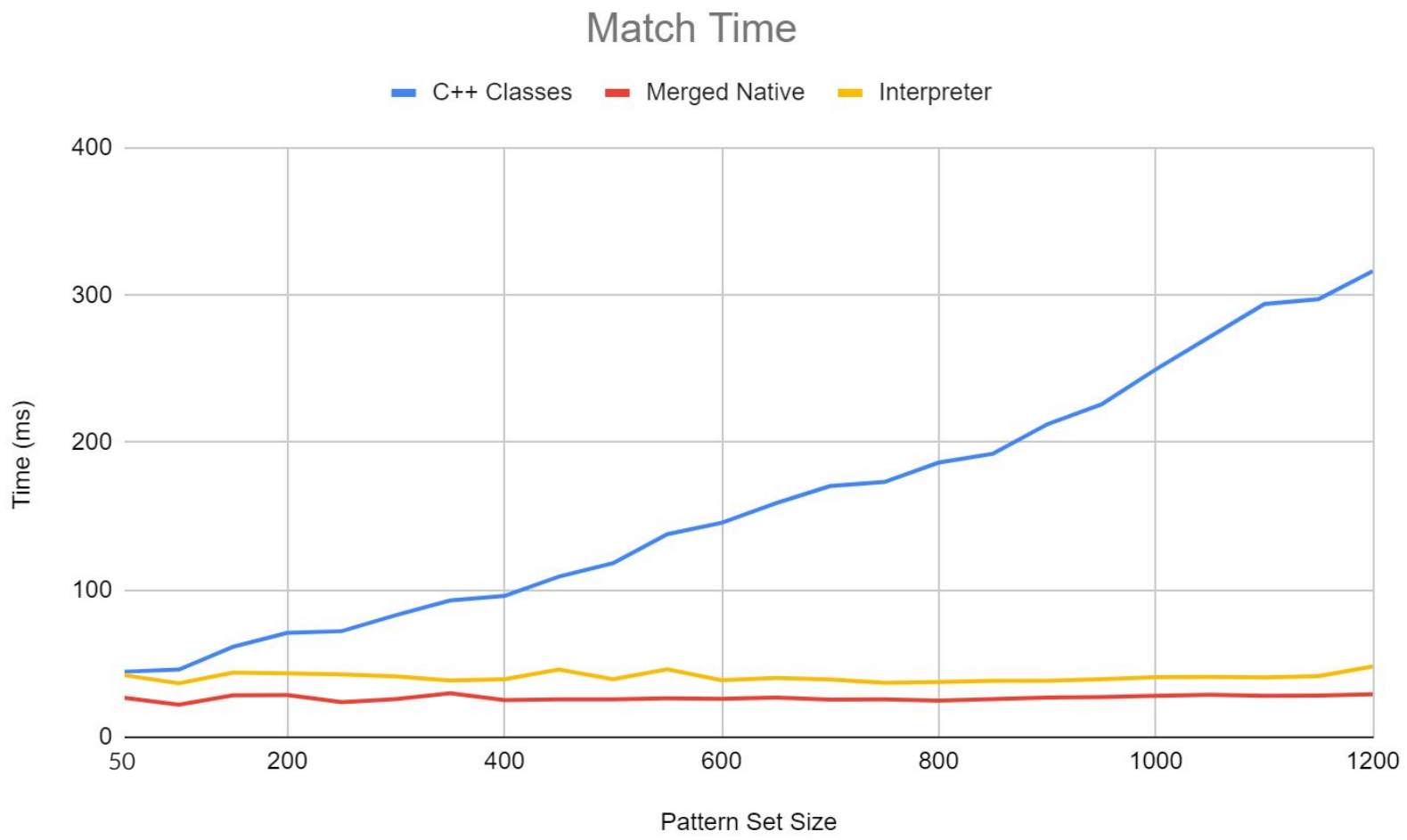


Metrics

Estimates*

- Bytecode is ~10x smaller than pattern object file
- Pattern merge time is ~15x faster than compiling in Clang

Pattern Match Runtime



Pattern Match Runtime

- Seems to remove number of patterns from runtime complexity
- Interpreter adds ~15 ms of overhead

Current Status

- Not upstream
- No support for ODS variadic arguments as in TableGen
- Otherwise feature-complete
- Hybrid pattern generator from TableGen

Next Steps

- Not upstream
- No support for ODS variadic arguments as in TableGen
- Otherwise feature-complete
- Hybrid pattern generator from TableGen

That's all folks

Questions?