Practical Second Futamura Projection

Partial Evaluation for High-Performance Language Interpreters

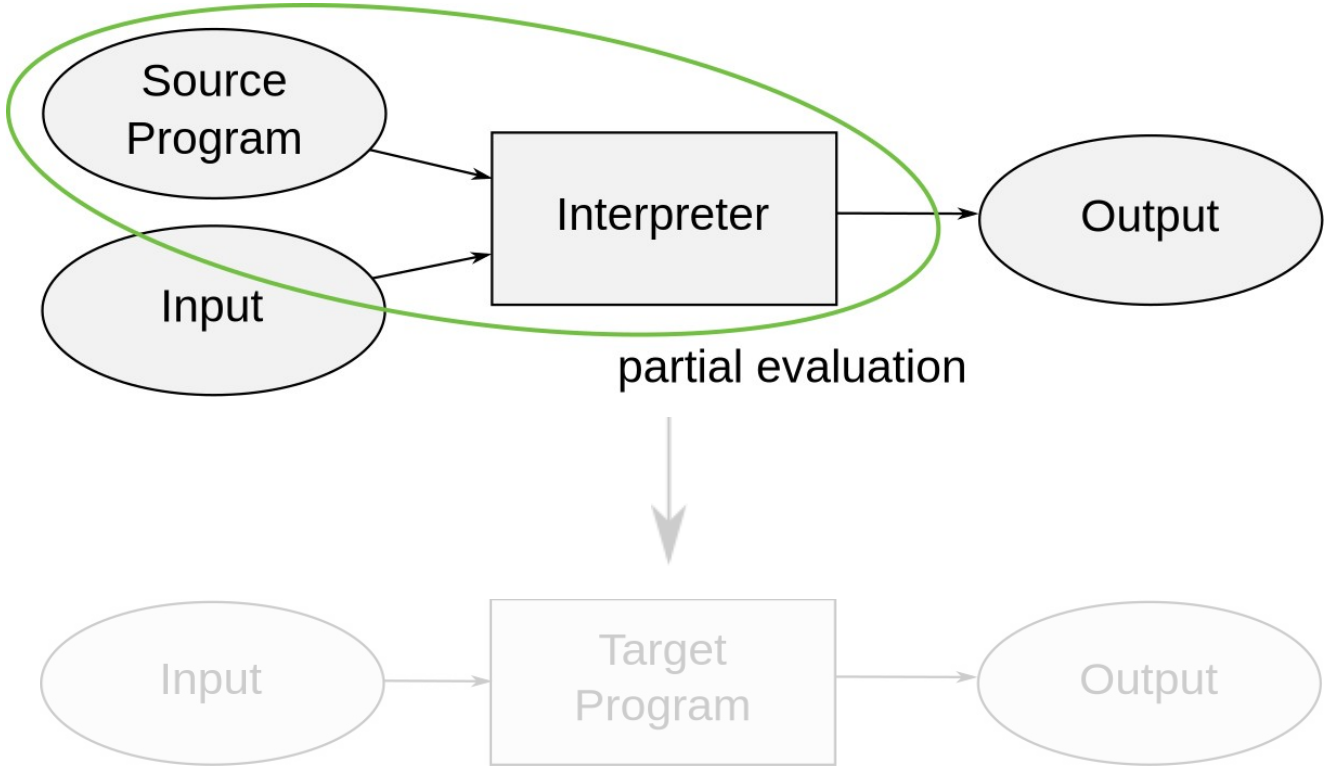
Florian Latifi

SPLASH 2019

Doctoral Symposium – Main Talk

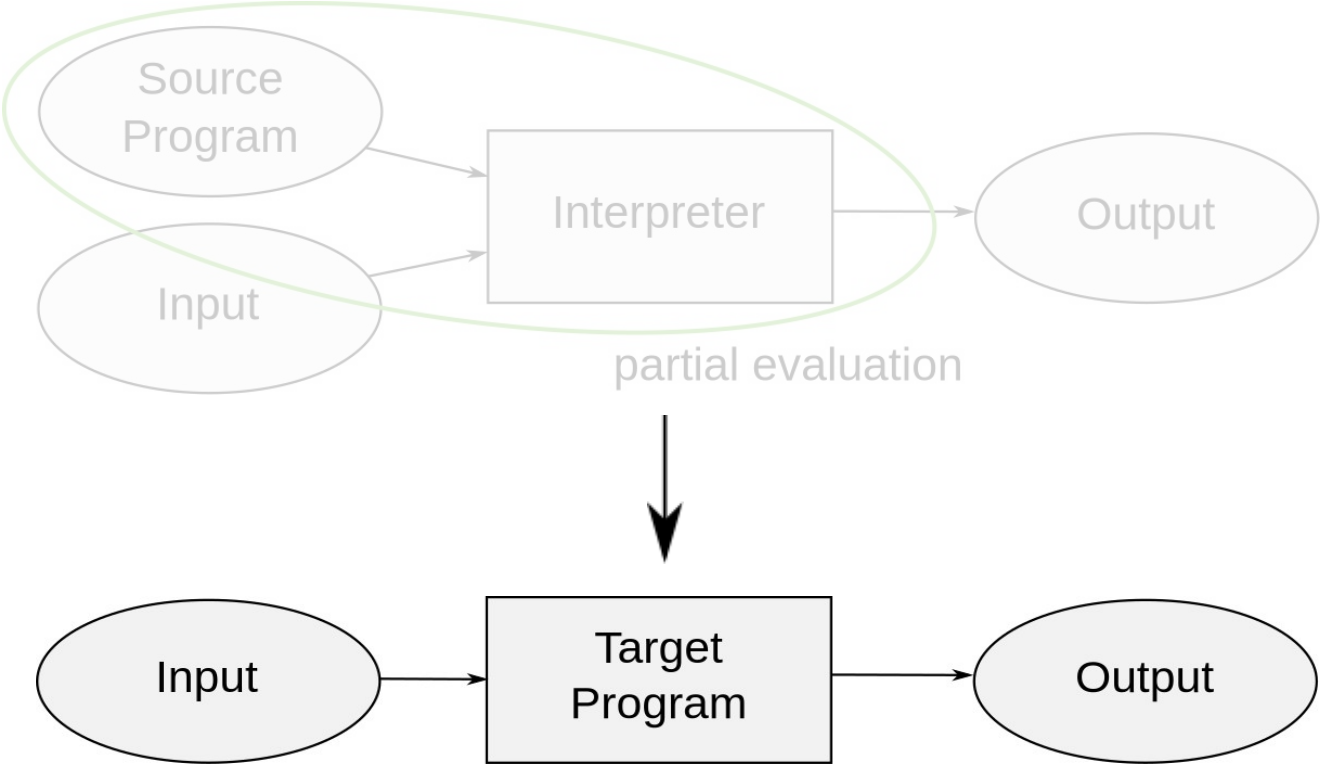
First Futamura Projection

- Specialize an interpreter for a given source program
- Yields an efficient & executable target program



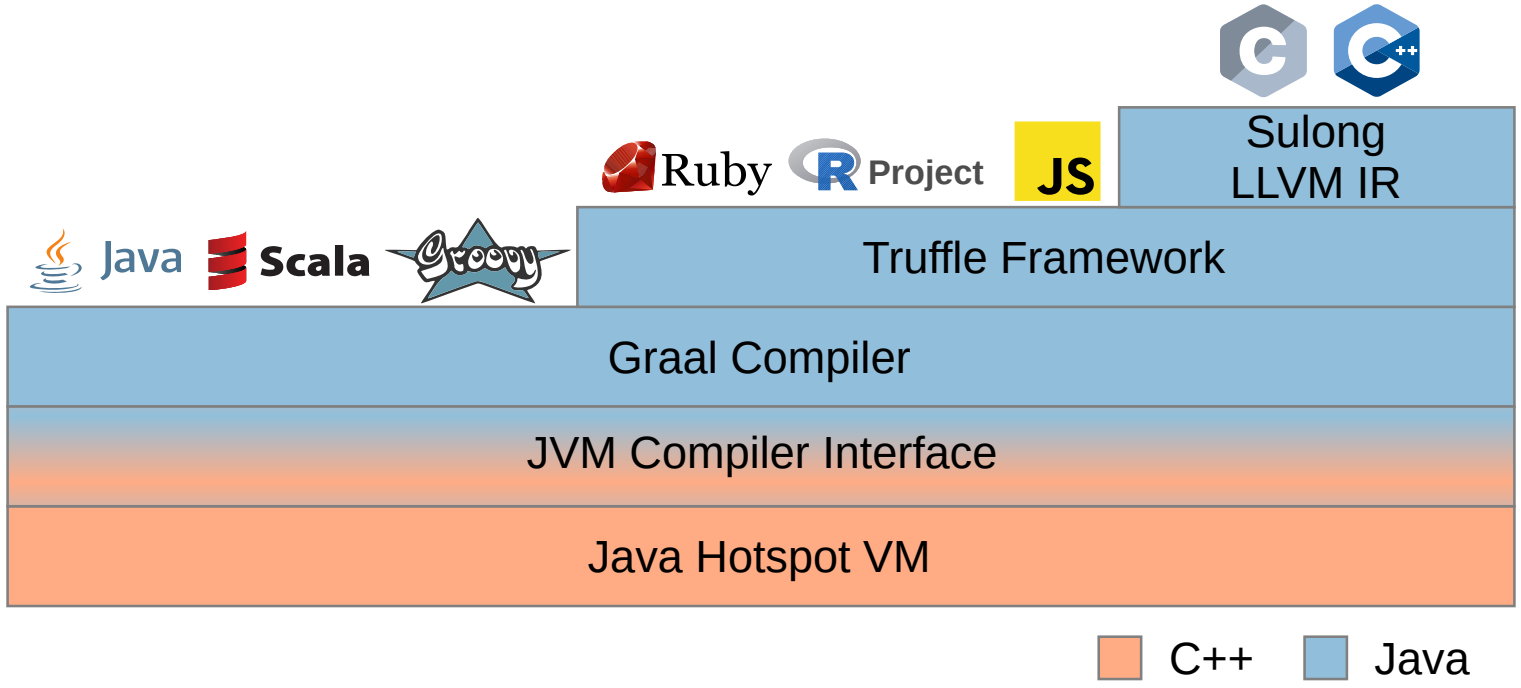
First Futamura Projection

- Specialize an interpreter for a given source program
- Yields an efficient & executable target program

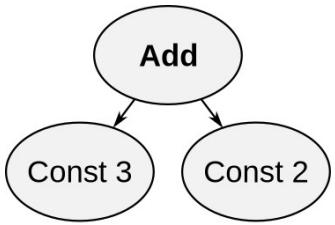


GraalVM

- Graal: dynamic optimizing compiler
- Truffle: language implementation framework



Example

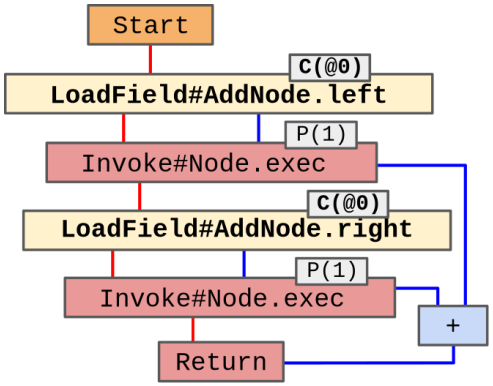


Example AST

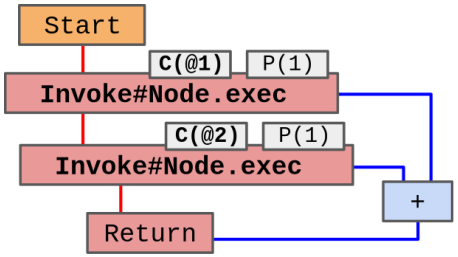
```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

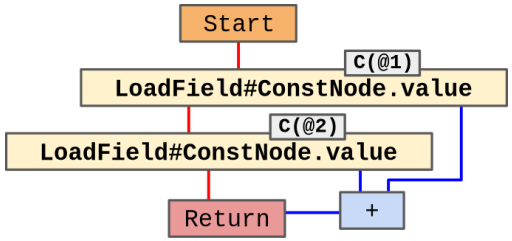
1) parse exec of root



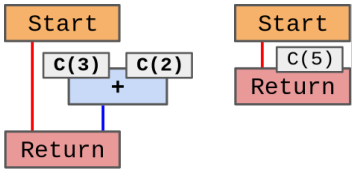
2) fold field loads assuming `@Child` fields are constant



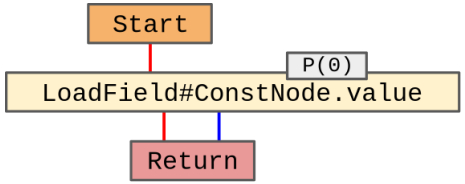
3) parse exec of `ConstNode` and inline invokes



4) fold loads of final fields



5) fold addition



Example

Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

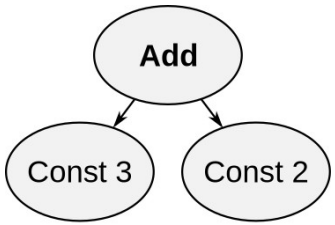
2) fold field loads assuming `@Child` fields are constant

3) parse exec of `ConstNode` and inline invokes

4) fold loads of final fields

5) fold addition

Example



Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

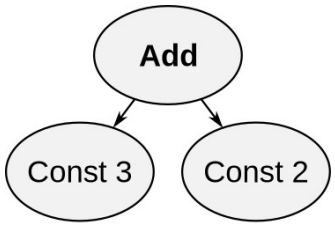
2) fold field loads assuming `@Child` fields are constant

3) parse exec of `ConstNode` and inline invokes

4) fold loads of final fields

5) fold addition

Example



Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

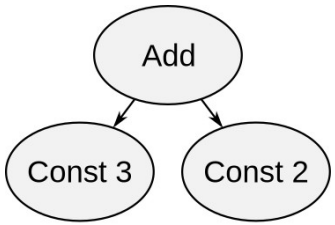
2) fold field loads assuming `@Child` fields are constant

3) parse exec of `ConstNode` and inline invokes

4) fold loads of final fields

5) fold addition

Example

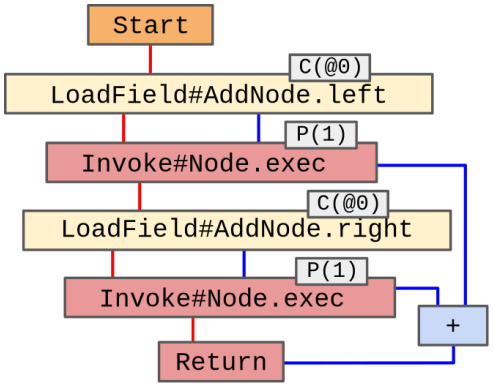


Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root



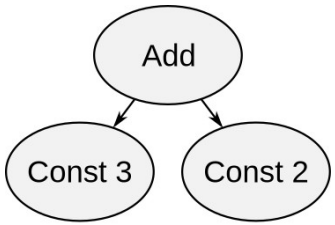
2) fold field loads assuming `@Child` fields are constant

3) parse exec of `ConstNode` and inline invokes

4) fold loads of final fields

5) fold addition

Example

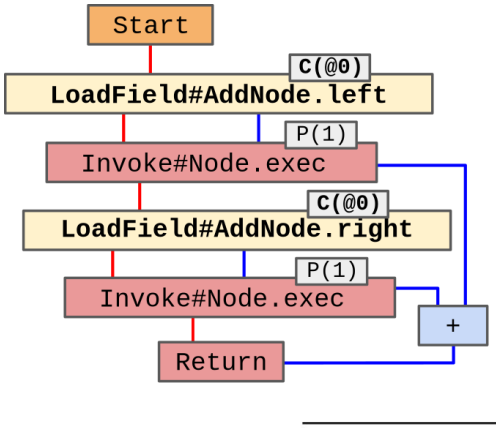


Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root



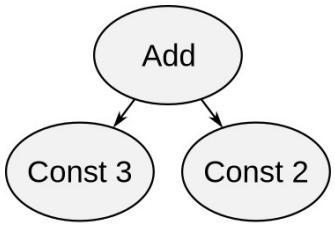
2) fold field loads assuming `@Child` fields are constant

3) parse exec of `ConstNode` and inline invokes

4) fold loads of final fields

5) fold addition

Example



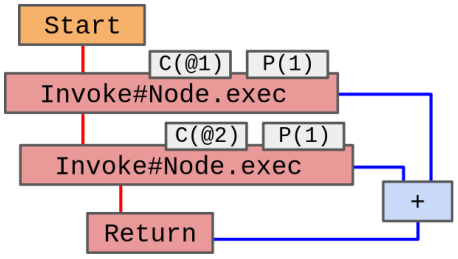
Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

2) fold field loads assuming `@Child` fields are constant

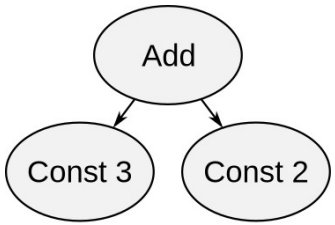


3) parse exec of `ConstNode` and inline invokes

4) fold loads of final fields

5) fold addition

Example



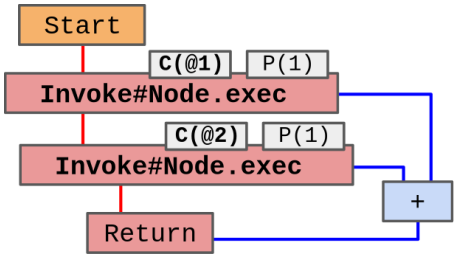
Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

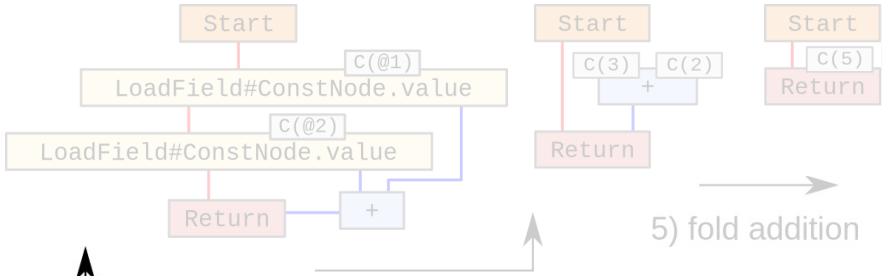
```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

2) fold field loads assuming `@Child` fields are constant



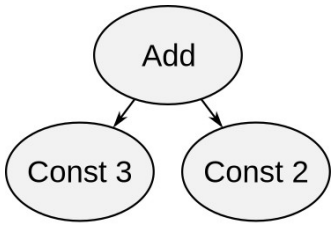
3) parse exec of `ConstNode` and inline invokes



4) fold loads of final fields

5) fold addition

Example



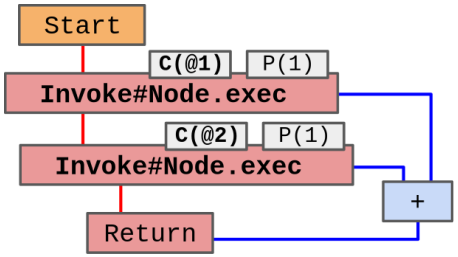
Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

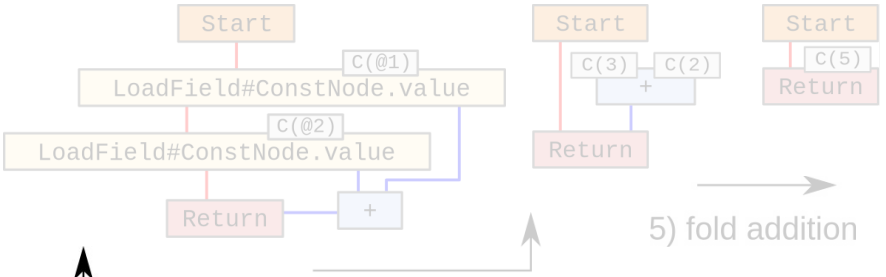
```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

2) fold field loads assuming `@Child` fields are constant

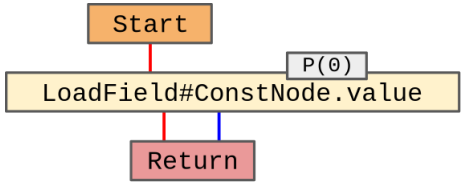


3) parse exec of `ConstNode` and inline invokes

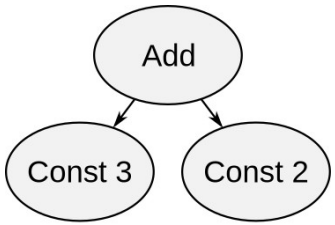


4) fold loads of final fields

5) fold addition



Example



Example AST

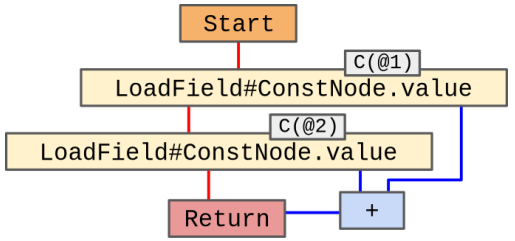
```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

2) fold field loads assuming `@Child` fields are constant

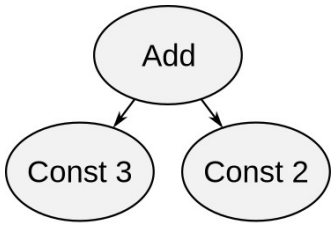
3) parse exec of `ConstNode` and inline invokes



4) fold loads of final fields

5) fold addition

Example



Example AST

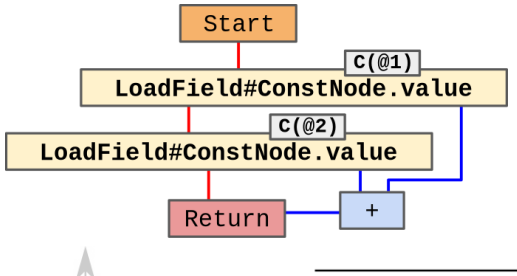
```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

2) fold field loads assuming `@Child` fields are constant

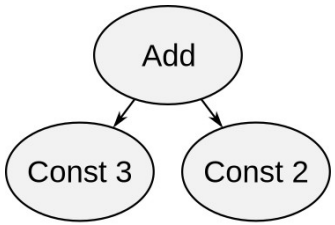
3) parse exec of `ConstNode` and inline invokes



4) fold loads of final fields

5) fold addition

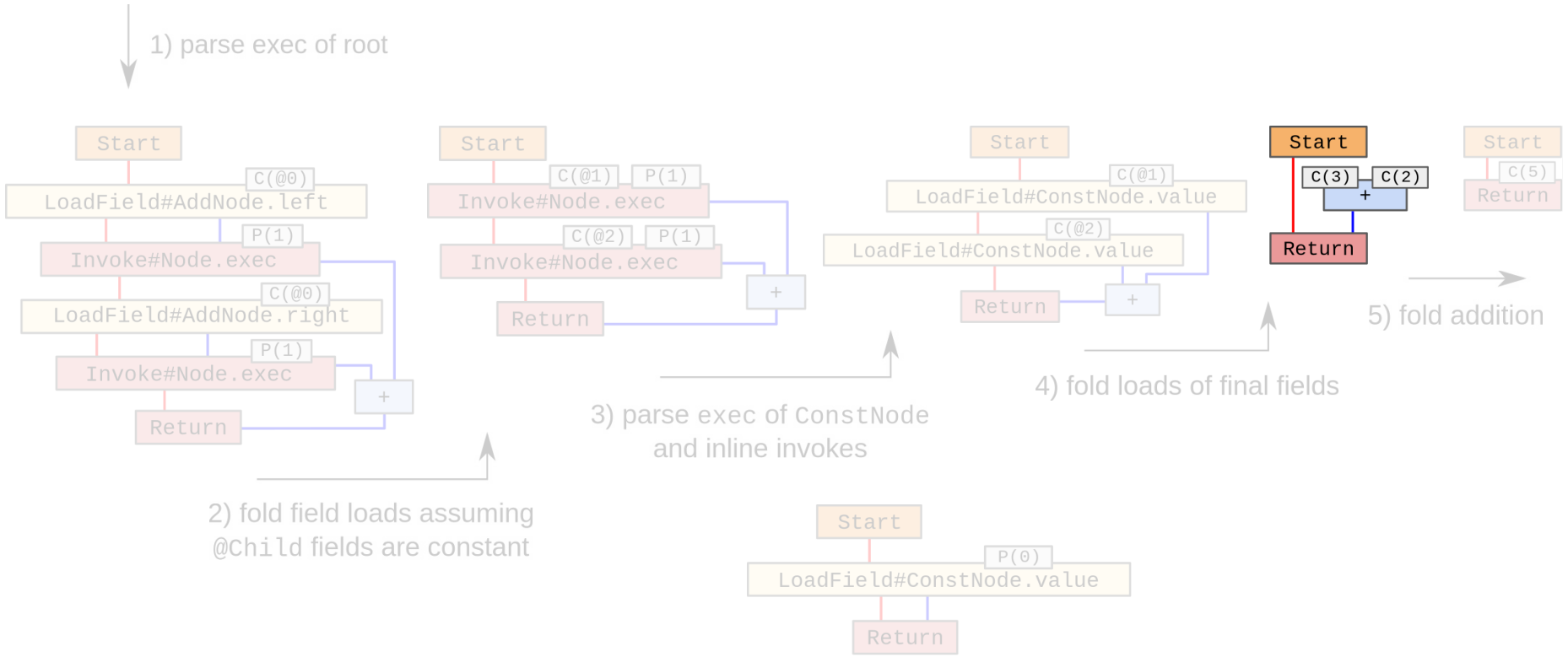
Example



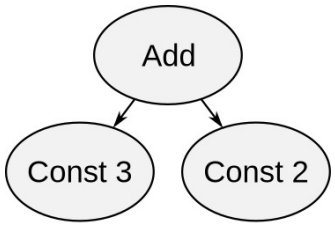
Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```



Example



Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

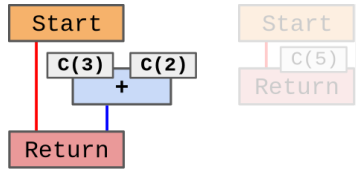
```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

2) fold field loads assuming `@Child` fields are constant

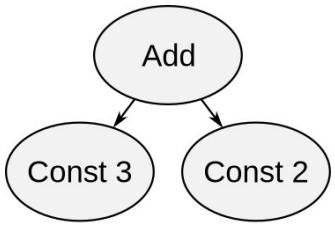
3) parse exec of `ConstNode` and inline invokes

4) fold loads of final fields



5) fold addition

Example



Example AST

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

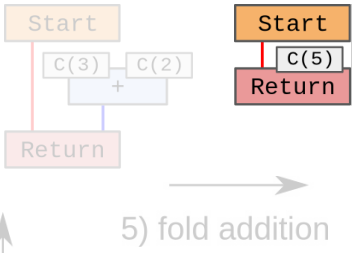
```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

1) parse exec of root

2) fold field loads assuming `@Child` fields are constant

3) parse exec of `ConstNode` and inline invokes

4) fold loads of final fields



5) fold addition

Problem

- Partial evaluation is compile-time intensive for large ASTs
- Overall compile time:
  - ~50% partial evaluation
  - ~50% optimization & machine code generation

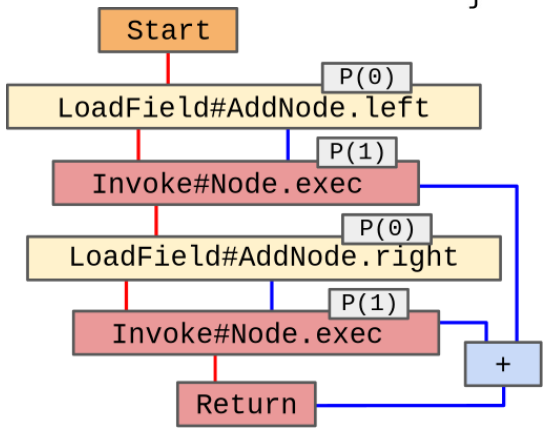
Goal

- Speed up partial evaluation
- With this, reduce overall compile time

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

parse

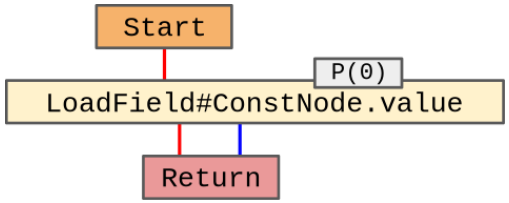



```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse




```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

JKU

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

parse


```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse


```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

parse


```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse

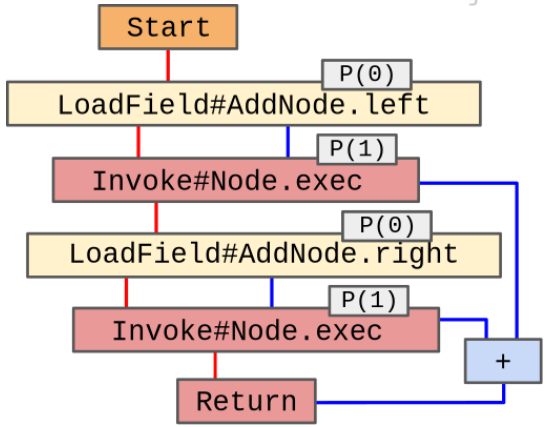

```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

parse




```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse

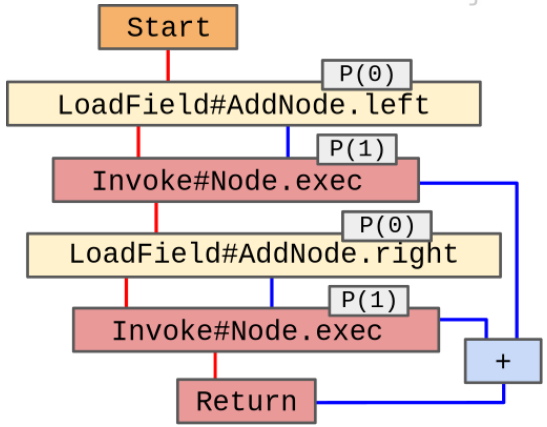

```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

parse




```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse


```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

parse


```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse


```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

parse


```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse


```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

JKU

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

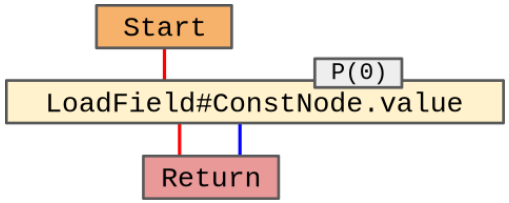
parse


```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse




```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

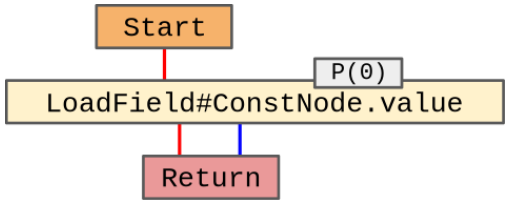
parse


```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse




```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

Approach

```
class AddNode extends Node {
 @Child Node left;
 @Child Node right;

 int exec(VirtualFrame f) {
  return left.exec(f) + right.exec(f);
 }
}
```

parse


```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

iterate in reverse post-order
and generate snippets

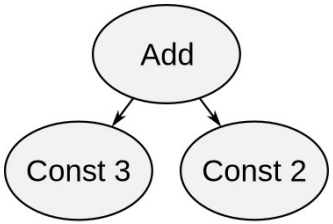
```
class ConstNode extends Node {
 final int value;

 int exec(VirtualFrame f) {
  return value;
 }
}
```

parse


```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

iterate in reverse post-order
and generate snippets

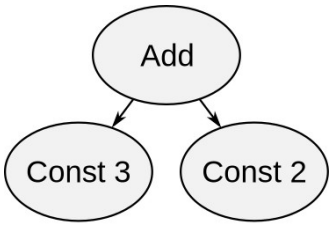
Example



Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

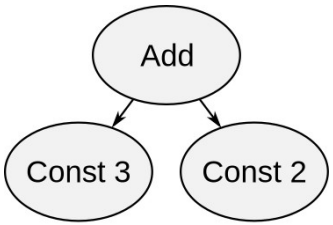
Example



Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

Example



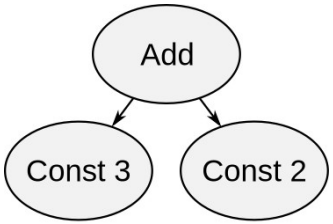
Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph

Start

Example



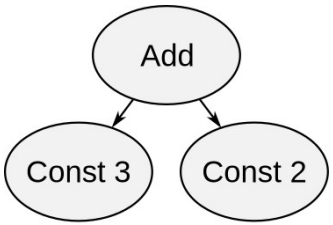
Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph args[0] args[1]

Start

Example



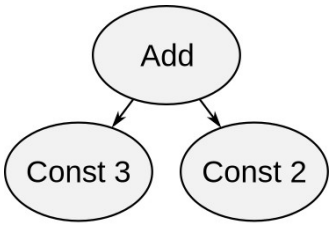
Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph | args[0] | args[1]

Start | C(@0)

Example

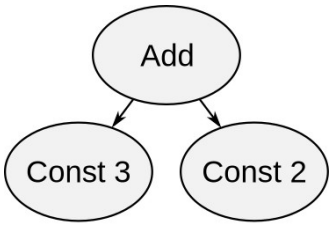


Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph	args[0]	args[1]
Start	C(@0)	P(1)

Example



Example AST

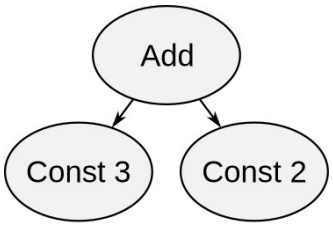
```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph	args[0]	args[1]
Start	C(@0)	P(1)

```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

args[0]	args[1]
C(@0)	P(1)

Example



Example AST

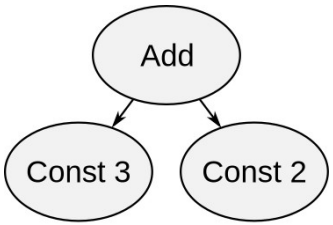
```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph	args[0]	args[1]
Start	C(@0)	P(1)

```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

args[0]	args[1]	left
C(@0)	P(1)	C(@1)

Example



Example AST

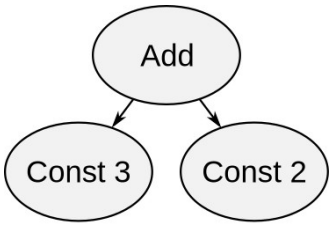
```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph	args[0]	args[1]
Start	C(@0)	P(1)

```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

args[0]	args[1]	left	right
C(@0)	P(1)	C(@1)	C(@2)

Example



Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph	args[0]	args[1]
Start	C(@0)	P(1)

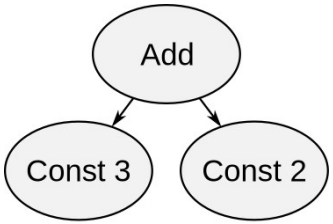
```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

args[0]	args[1]	left	right
C(@0)	P(1)	C(@1)	C(@2)

```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

args[0]	args[1]
C(@1)	P(1)

Example



Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

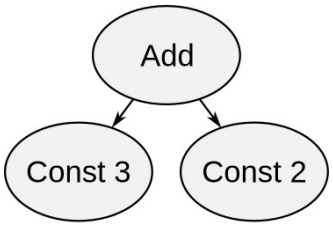
graph	args[0]	args[1]
Start	C(@0)	P(1)

```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

args[0]	args[1]	left	right
C(@0)	P(1)	C(@1)	C(@2)

leftVal
C(3)

Example



Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

graph	args[0]	args[1]
Start	C(@0)	P(1)

```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

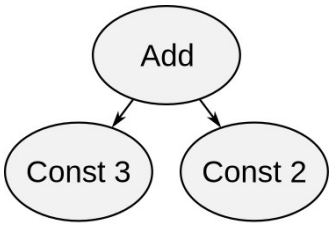
args[0]	args[1]	left	right
C(@0)	P(1)	C(@1)	C(@2)

leftVal
C(3)

```
IRNode generate_ConstNode_exec(IRNode... args) {
 return loadFieldAsConst(args[0], "value");
}
```

args[0]	args[1]
C(@2)	P(1)

Example



Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

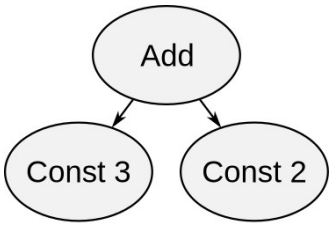
graph	args[0]	args[1]
Start	C(@0)	P(1)

```
IRNode generate_AddNode_exec(IRNode... args) {
 IRConst left = loadFieldAsConst(args[0], "left");
 IRConst right = loadFieldAsConst(args[0], "right");
 IRNode leftVal = dispatch("Node#exec", left, args[1]);
 IRNode rightVal = dispatch("Node#exec", right, args[1]);
 return AddNode.create(leftVal, rightVal);
}
```

args[0]	args[1]	left	right
C(@0)	P(1)	C(@1)	C(@2)

leftVal	rightVal
C(3)	C(2)

Example

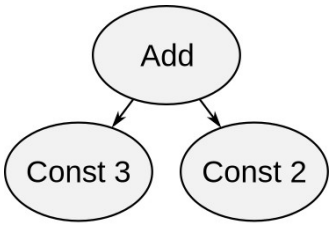


Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

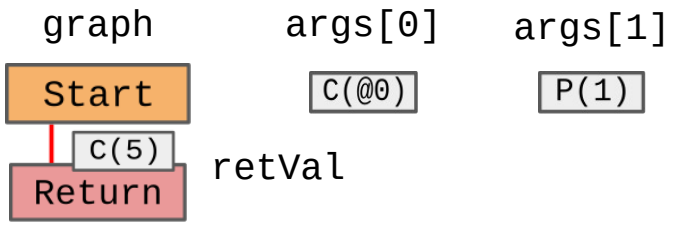
graph	args[0]	args[1]	retVal
Start	C(@0)	P(1)	C(5)

Example



Example AST

```
IRGraph doPE2(Node root) {
 IRGraph graph = IRGraph.create();
 IRNode args = new IRNode[2];
 args[0] = IRConst.forObject(root);
 args[1] = IRParam.forIndex(1);

 IRNode retVal = dispatch("Node#exec", args);
 finishGraph(graph, retVal);
 return graph;
}
```

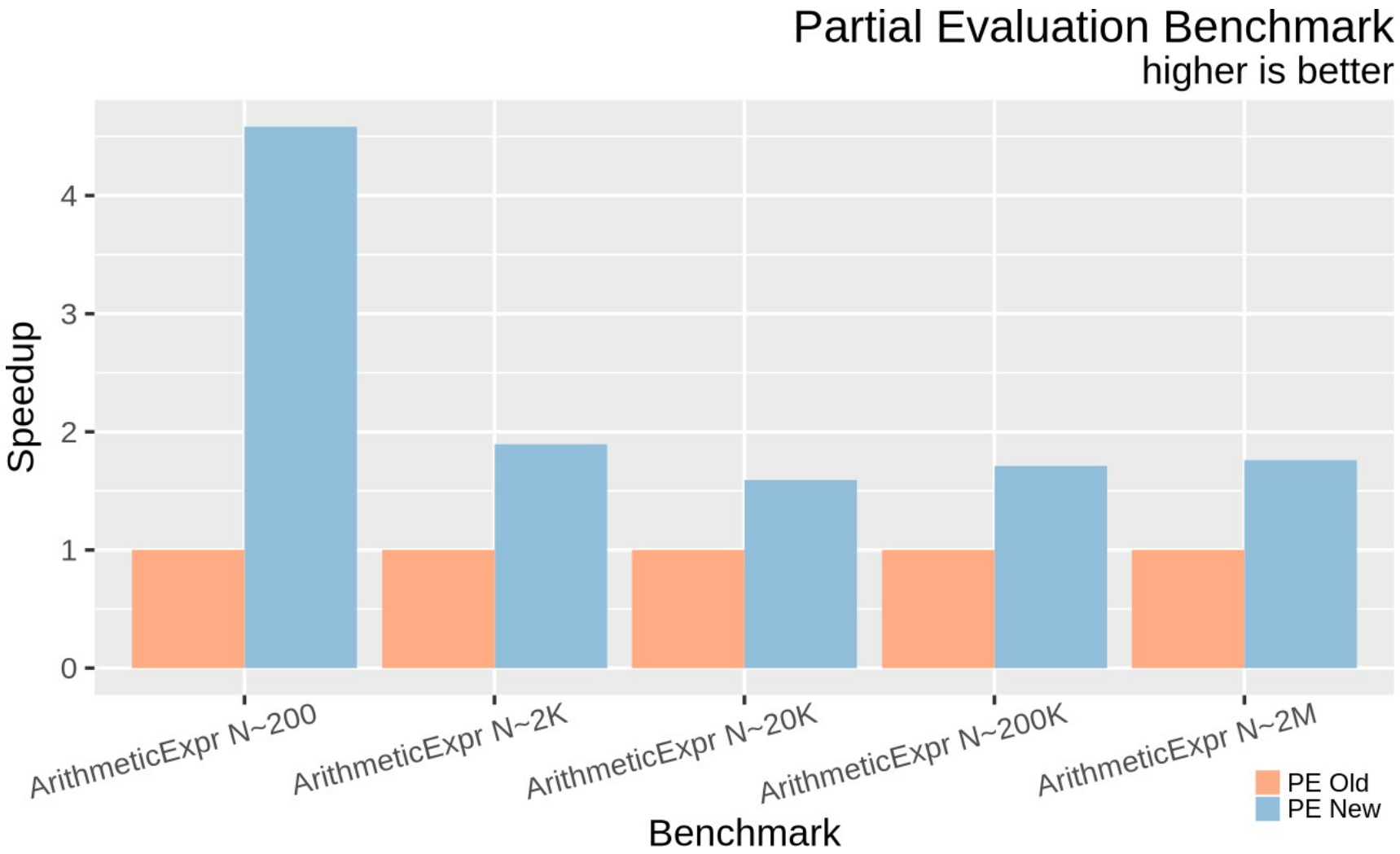


Evaluation Methodology

- Comparison:
  - existing partial evaluation mechanism
  - generated partial evaluation snippets

- Measurements:
  - partial evaluation and overall compile time
  - code-size increase due to code generation
- Benchmarks:
  - common language benchmarks for Truffle implementations
  - Graal.js for JavaScript, Sulong for LLVM-IR, …

First Results



QA