PHP Generators in .NET

Compilation of a dynamic langauge Generators into MSIL

FACULTY OF MATHEMATICS AND PHYSICS — Charles University

PEACHPIE

Petr Houška | Mgr. Jakub Míšek | Department of Software engeneering | github.com/peachpiecompiler/peachpie | github.com/petrroll/bachelor-thesis

Goals

- Generators support within Peachpie
- Keeping reference PHP semantics
- Reusing already existing facilities
- Both design and implementation

Peachpie

- Bridge between PHP and .NET
- PHP to MSIL (CIL) compiler
- Reimplementation of PHP class library
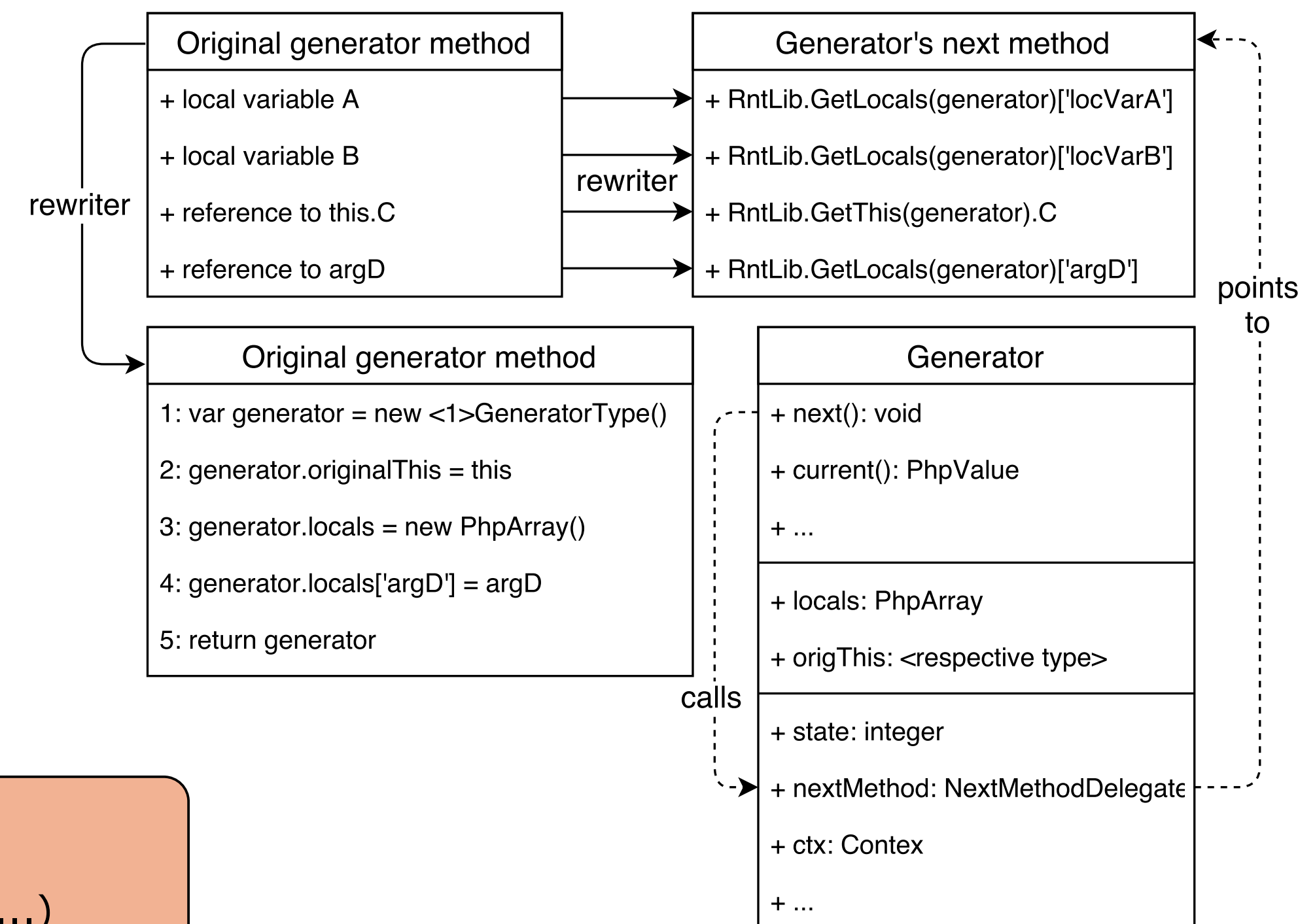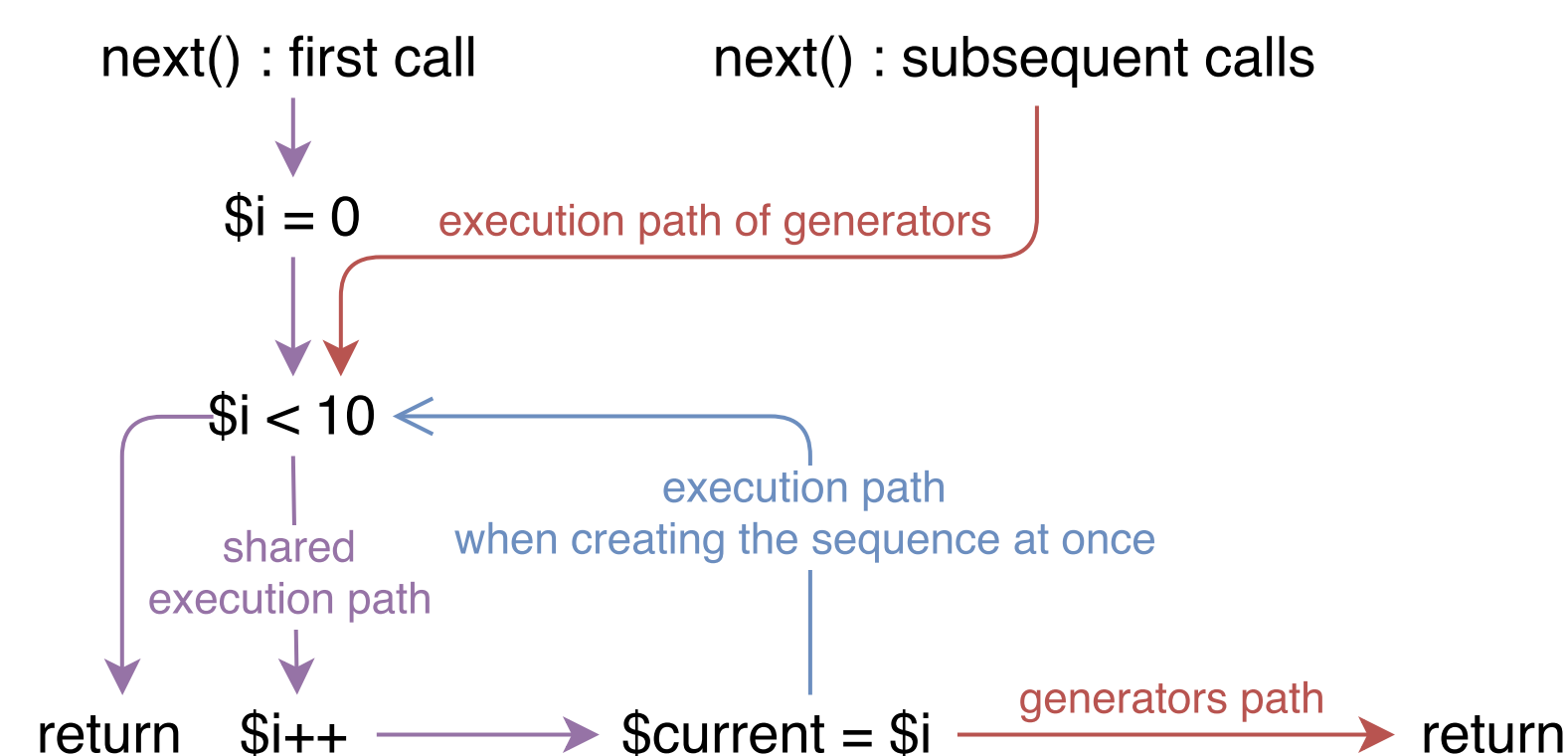- Written in C#, .NET foundation member

MSIL

- Intermediate language for .NET platform (C#, F#, ...)
- High level, stack based, object oriented assembler
- Native understanding of exception handling
- No concept of generators or execution state saving

Generators

- Contain a sequential alghoritm, return an Iterator
- Natively supported by the reference PHP runtime (Zend)
- Lowered by compilers in .NET languages (e.g. Roslyn)
- Execution pauses at *yield* keyword, continues on *next()* call
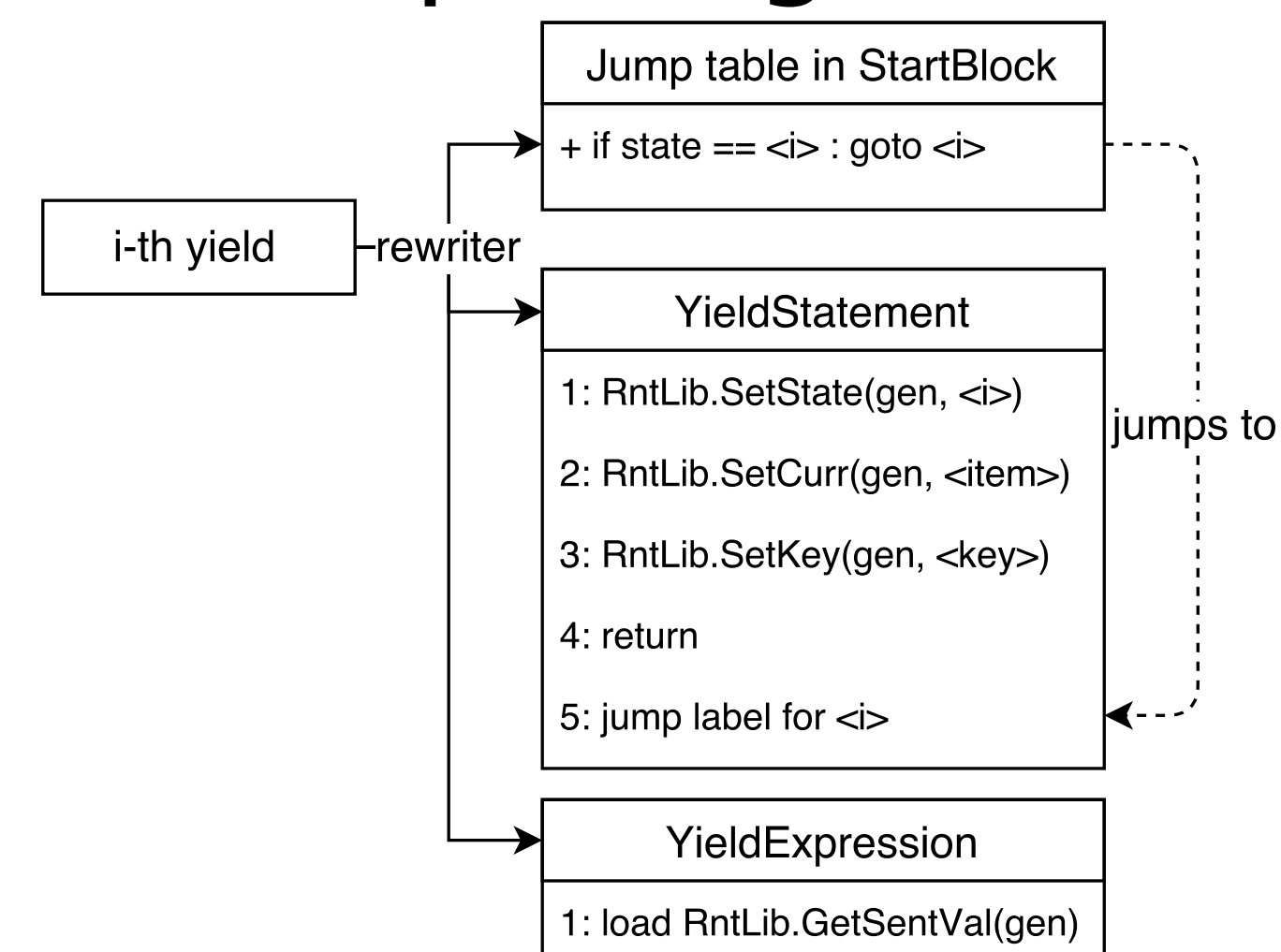 - Statement in most .NET languages (C#, F#)
  - Value carying expression in PHP

Generator method


```
next() : first call          next() : subsequent calls

$i = 0          execution path of generators

$i < 10          execution path
                 when creating the sequence at once

shared
execution path

return   $i++   →   $current = $i   generators path   return
```

Original generator method

Original generator method
+ local variable A
+ local variable B
+ reference to this.C
+ reference to argD

rewriter →

Generator's next method
+ RntLib.GetLocals(generator)['locVarA']
+ RntLib.GetLocals(generator)['locVarB']
+ RntLib.GetThis(generator).C
+ RntLib.GetLocals(generator)['argD']

points to

Original generator method
1: var generator = new <1>GeneratorType()
2: generator.originalThis = this
3: generator.locals = new PhpArray()
4: generator.locals['argD'] = argD
5: return generator

calls

Generator
+ next(): void
+ current(): PhpValue
+ ...
+ locals: PhpArray
+ origThis: <respective type>
+ state: integer
+ nextMethod: NextMethodDelegate
+ ctx: Contex
+ ...

Generator method

```
function by_one_generator(){
  for($i = 0; $i < 10; $i++){
    yield $i;
  }
}
```

Yield splitting

Jump table in StartBlock
+ if state == <i> : goto <i>

i-th yield — rewriter →

YieldStatement
1: RntLib.SetState(gen, <i>)
2: RntLib.SetCurr(gen, <item>)
3: RntLib.SetKey(gen, <key>)
4: return
5: jump label for <i>

jumps to

YieldExpression
1: load RntLib.GetSentVal(gen)

Solution

- New Generator type representing the returned Iterator
- Changes to the original generator method's body:
  - Transformed to Generator's state machine as a new method
  - Replaced with code that initiates and returns the Generator
- State machine transformations:
  - Local variables lifted to Generator instance
  - Exlicit state saving on each yield
  - State backed jump table in the beginning
- Semantic tree transformation to lower yields:
  - Moves yields under expression trees' roots
  - Handles conditioned branches with yields
  - Maintains the original order of execution
  - Splitting yield into a statement and an expression

Semantic tree transformation