

PHP Generators in .NET PEACHPIE Compilation of a dynamic langauge Generators into MSIL



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

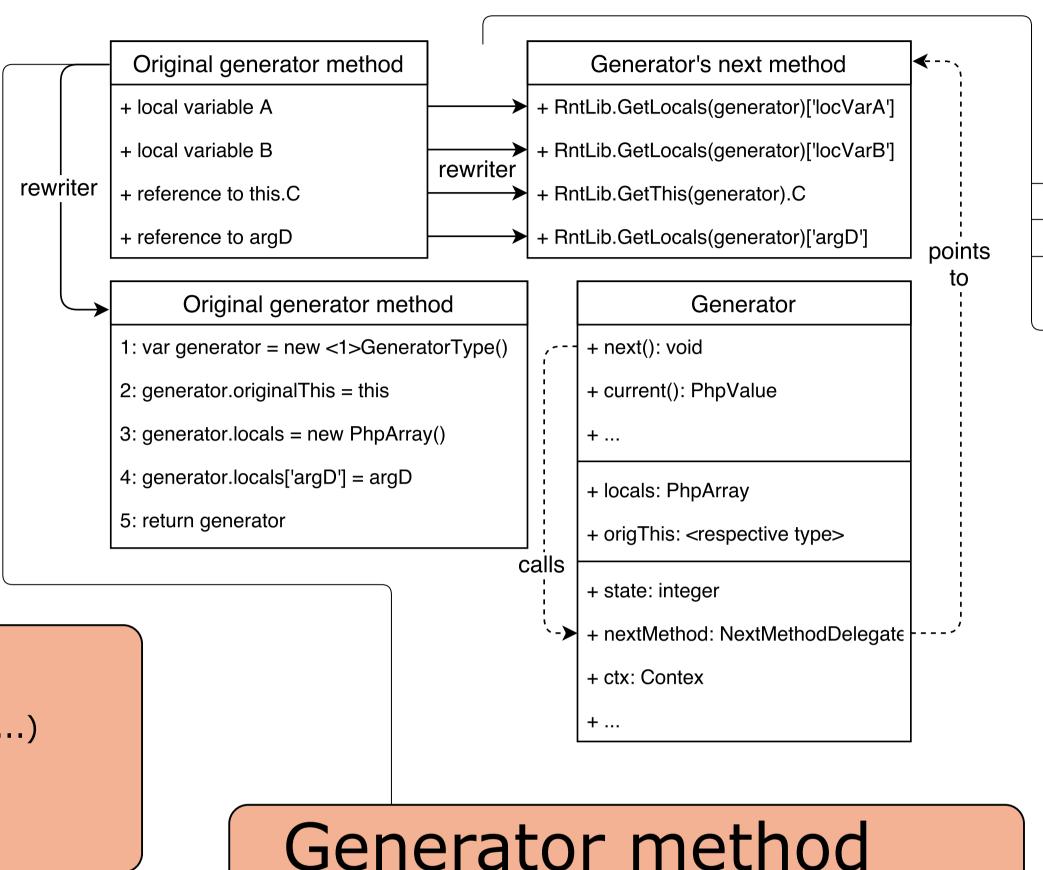
MSIL

- 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 execution path return \$i++ ---



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>) iumps to 2: RntLib.SetCurr(gen, <item>) 3: RntLib.SetKey(gen, <key>) 4: return 5: jump label for <i> YieldExpression 1: load RntLib.GetSentVal(gen)

Solution

- New Generator type representing the returned Iterator
- Changes to the original generator method's body:
- Transforms it to Generator's state machine as a new method
- Replaces it with code that initiates and returns the Generator
- State machine transformations:
- Lifts local variables to a Generator instance
- Explicitly saves state on each yield
- Adds a state backed jump table in the beginning
- Semantic tree transformation to lower yields:
- Splits yields into a statement and an expression
- Moves yields under expression trees' roots
- Handles conditioned branches with yields

- Maintains the original order of execution

Semantic tree transformation Expr. 5 Expr. 1 Yield Expr. Expr. 4 Expr. 2 capture of yield branch **YieldStatement** Assignment statement Expr. 1 directly followed by Variable 'tmp<1>' expr. Assignment statement **Assignment statement** YieldExpression Expr. 4 Variable Variable 'tmp<2>' expr. 'tmp<3>' expr. Expr. 2 Original statment Assignment statement Variable Expr. 5 Variable 'tmp<4>' expr. 'tmp<4>' expr. Variable Variable 'tmp<1>' expr. 'tmp<3>' expr. **Variable** 'tmp<2>' expr.