Amaru - A Framework for Combining Genetic Improvement with Pattern Mining (amaru.dev)

Boston / Online 9. July 2022

GI@GECCO 2022
What is a pattern?

Abstract Syntax Tree (AST)
What is a pattern?

Many Abstract Syntax Trees (ASTs)

fn sum()
{}
write int arr[]
read arg 0 for
write int i
0 <
read int
i +
write int

fn min()
{}
write int arr[]
read arg 0 for
write int i
0 <
read int
i +
write int

fn max()
{}
write int arr[]
read arg 0 for
write int i
0 <
read int
i +
write int

fn avg()
{}
write int arr[]
read arg 0 for
write int i
0 <
read int
i +
write int
What is a pattern?

Many Abstract Syntax Trees (ASTs)

```
fn sum()
{}
write int
arr[]
read arg 0
for
write int i
0
<
read int i
read int len
write int ... i
0
<
read int i
read int len
write int i
+
read int i 1
```

Frequent Pattern
Why Mine Patterns?

- Software engineering is challenging

Why Mine Patterns?

- Software engineering is challenging
- More challenging when needing to optimize for Non-Functional Properties (NFP)
Why Mine Patterns?

- Software engineering is challenging
- More challenging when needing to optimize for Non-Functional Properties (NFP)
- Example: 75% of Software Maintenance costs improve performance or fix bugs [1]

Why Mine Patterns?

- Software engineering is challenging
- More challenging when needing to optimize for **Non-Functional Properties (NFP)**
- Example: 75% of Software Maintenance costs improve performance or fix bugs [1]
- Goal: Find patterns, validate patterns, use patterns

Introduction I

- Genetic Improvement
  - provide data
  - improve with patterns
  - representation, language access

- Pattern Mining
  - patterns

- Compiler and Interpreter
  - GraalVM
  - Truffle
The Amaru Framework

Execution Environment
- Guest Application
- Guest Language
- Truffle API
- Graal VM
- Java HotSpot VM

Optimization Framework & API
- source code & AST
- Experiment
  - Configuration
  - Test Cases
  - Original AST
  - Optimized AST
- Experiment Results
  - Tested AST
  - Test Results
  - Runtime Profile

Knowledge Base
- Language Information
- Node Information
- Original AST
- Optimized AST
- Transformation Patterns

Pattern Framework & API
- Truffle Language Analyzer (TLA)
- Truffle Language Information (TLI)
  - Truffle Nodes
    - Terminals
    - Non-Terminals
  - Initialization Mechanism

Key:
- Execution Relation
- Data Relation
- Logical Group
- Used Framework
- Language Specific
- Program Specific
- Data
- Optional Data
The Amaru Framework

**Execution Environment**
- Graal VM
- Java HotSpot VM

**Pattern Framework & API**
- Truffle Language
  - Information (TLI)
- Truffle Language Analyzer (TLA)

**Optimization Framework & API**
- Experiment
  - Configuration
  - Test Cases
  - Optimal AST
  - Original AST
- Truffle Language Information (TLI)
  - Truffle Nodes
    - Terminals
    - Non-Terminals
  - Initialization Mechanism
- Truffle Pattern Injector (TPI)
  - Pattern Application Algorithms
    - collected AST
    - transformed AST
- Truffle Pattern Detector (TPD)
  - Pattern Detection Algorithms
    - Original AST
    - Optimized AST

**Knowledge Base**
- Language Information
- Node Information
  - Tested AST
    - Test Results
    - Runtime Profile

**Key:**
- Execution Relation
- Data Relation
- Logical Group
- Used Framework
- Language Specific
- Program Specific
- Data
- Optional Data
The Amaru Framework

- **Run directly in compiler**
  - + Stack/heap access
  - + Granularity
  - + Information on language
  - - Granularity
  - - Work needed for GI operators

- **Optimization Framework & API**
  - + Learn about language
  - + Apply knowledge in GI
  - + Apply patterns
  - - Performance of GI operators

- **Knowledge Base**
  - Language Information
  - Node Information
  - Original AST
  - Optimized AST

- **Transformation Patterns**
  - Truffle Pattern Detector (TPD)
    - Pattern Detection Algorithms
    - Pattern Application Algorithms
  - Truffle Pattern Injector (TPI)
    - collected AST
    - transformed AST

- **Optimizer**
  - Connector to other Frameworks
  - Optimization Algorithms

- **Experiment Results**
  - Tested AST
    - Test Results
    - Runtime Profile

- **Key**
  - Execution Relation
  - Data Relation
  - Logical Group
  - Used Framework
  - Language Specific
  - Program Specific
  - Data
  - Optional Data

---

Slide 4 / 19
The Amaru Framework

Run directly in compiler
- Stack/heap access
- Granularity
- Information on language
  - Granularity
  - Work needed for GI operators

Algorithms that learn
- Learn about language
- Apply knowledge in GI
- Apply patterns
- Performance of GI operators

Knowledge base
+ Store experiment data for analysis
+ Reproducible experiments
+ Publish experiment data

Key: 
- Execution Relation
- Data Relation
- Logical Group
- Used Framework
- Language Specific
- Program Specific
- Data
- Optional Data
The Amaru Framework

- Run directly in compiler
  + Stack/heap access
  + Granularity
  + Information on language
    - Granularity
    - Work needed for GI operators

- Optimization Framework & API
  source code & AST
  describes
  analyzes

- Algorithms that learn
  + Learn about language
  + Apply knowledge in GI
  + Apply patterns
    - Performance of GI operators

- Extensible algorithms
  + Connect to other frameworks
  + Parallel / distributed execution

- Knowledge base
  + Store experiment data for analysis
  + Reproducible experiments
  + Publish experiment data

Key:
- Execution Relation
- Data Relation
- Logical Group
- Used Framework
- Language Specific
- Program Specific
- Data
- Optional Data
The Amaru Framework

**Run directly in compiler**
- Stack/heap access
- Granularity
- Information on language
  - Granularity
  - Work needed for GI operators

**Optimization Framework & API**
- Source code & AST
  - Analyses
  - Describes
  - Uses

**Algorithms that learn**
- Learn about language
- Apply knowledge in GI
- Apply patterns
  - Performance of GI operators

**Extensible algorithms**
- Connect to other frameworks
- Parallel / distributed execution

**Pattern mining**
- Identify patterns
- Validate patterns

**Knowledge base**
- Store experiment data for analysis
- Reproducible experiments
- Publish experiment data

<table>
<thead>
<tr>
<th>Key:</th>
<th>Execution Relation</th>
<th>Data Relation</th>
<th>Logical Group</th>
<th>Used Framework</th>
<th>Language Specific</th>
<th>Program Specific</th>
<th>Data</th>
<th>Optional Data</th>
</tr>
</thead>
</table>

**JU**
Johannes Kepler Universität Linz
Knowledge-guided Genetic Improvement

Genetic Improvement
Knowledge-guided Genetic Improvement (KGGI)

provide data
improve with patterns
representation, language access

Pattern Mining

patterns

Compiler and Interpreter

GraalVM
Truffle
Knowledge-guided Genetic Improvement

- Grammar-guided Genetic Programming
- Tree Genetic Programming
- Enriched with metadata
  - loops, branches, NFP, ...
- Operators access context
  - stack, heap, functions, ...
- Applies patterns and requirements via Syntax Graph
Knowledge-guided Genetic Improvement

**AST**

```
...  
write len  
  10  
for  
write i  
  0  
MUTATE  
()  
...  
```

**Syntax Graph**

```
Entry Strategy  
  Root Strategy  
    <  
    >  
read int  
if  
...  
```
Knowledge-guided Genetic Improvement

Constraints:
- depth: 3
- branches: 1
- approx. perf: 100ns

Syntax Graph:
- Entry Strategy
- Root Strategy
- <
- >
- read int
- if
- ...

AST:
- write len
- 10
- for
- write i
- MUTATE
- {}
- 0
- ...

Slide 7 / 19
Knowledge-guided Genetic Improvement

**AST**

- `write len`
- `10`
- `for`
- `write i`
- `0`
- `MUTATE`
- `

**Syntax Graph**

- `read int`
- `if`
- `

**Constraints**

**Grammar**
Knowledge-guided Genetic Improvement

**AST**

- `write len`
  - `10`
  - `write i`
    - `0`
    - `MUTATE`
  - `for`<
  - `<`
  - `write i`
    - `...`
- `for`
- `for`
- `for`
- `Constraints`
  - `Grammar`
  - `Patterns`

**Syntax Graph**

- `read int`
- `if`
- `...`

- `Entry Strategy`
- `Root Strategy`
Knowledge-guided Genetic Improvement

AST

Syntax Graph

Constraints
Grammar
Patterns
Context

Entry Strategy
Root Strategy

write
len

write
i

read
i

MUTATE

for

<

10

{}
Knowledge-guided Genetic Improvement

**AST**

```
AST

write len

10

for

write i

<

read i

read len

len

10

Patterns

Constraints

Grammar

Context

...)

Syntax Graph

Entry Strategy

Root Strategy

<

>

read int

if

...
Independent Growth of Ordered Relationships

- Genetic Improvement
- Pattern Mining
- Independent Growth of Ordered Relationships (IGOR)
- Compiler and Interpreter
- GraalVM
- Truffle
Independent Growth of Ordered Relationships

- Mining of *frequently* recurring substructures
- *Significant* if occurring with a minimum support
- *Discriminative* pattern mining
  - Often used for software fault mining
  - Mining in two groups - *succeeding* and *failing*
  - Discriminative pattern occurs more often in one group
Independent Growth of Ordered Relationships

ASTs from GI experiments

Fast ASTs

Slow ASTs

ASTs with Exception "Timeout"

Metrics:
50% significant
75% discriminative

IGOR

Top n patterns

Block

DTArrayRead

InvokeVoid

Read

DTAnd

Read

amount of occurrence

amount of trees
Example: Bug Pattern
Uninitialized Variables

Fault of omission:
variable read before assignment
Example: Bug Pattern
Uninitialized Variables

Fault of omission:
variable read before assignment

Taxonomies
read
read
stack
read
heap
read
stack
int
read
stack
str
...

Generalized

Specialized
Example: Bug Pattern
Uninitialized Variables

- Embedded
- write 0
- read 0
- Missing

Fault of omission:
variable read before assignment

Taxonomies

Wildcards
Example: Bug Pattern
Uninitialized Variables

Fault of omission: variable read before assignment

Write missing before read

Taxonomies

Wildcards

Ordered Patterns
Example: Bug Pattern
Uninitialized Variables

- Fault of omission: variable read before assignment
- Taxonomies
- Wildcards
- Ordered Patterns
- Variables Considered

Write and read to same var 0
Validating patterns

- Using *KGGI Syntax Graph*
- Create Mutants of *n* ASTs
- Validate confidence in hypothesis
  - x% have a speedup due to pattern
  - x% fail due to exception
- Side effects prevent 100% confidence
Pattern Validation - Bug

(cause 82.7% confidence)

(fix 94.27% confidence)
Using Patterns in GI

Figure: Top: population in GI without patterns; bottom: with patterns
Improvement of Run-time Performance

Run time (ns)

Rectified Linear Activation  Leaky Rectified Linear Activation  Sigmoid  Swish  Tanh  Fully Inlined NN  All Activation Functions

Baseline patterns

Slide 15 / 19
Application of patterns in GI

GI population

- Amount of ASTs overall doubled
- Amount of successful ASTs doubled
- Only 32.7% of ASTs with exceptions (down from 60.3%)

Run-time performance

- 22 / 25 ASTs improved
- Average of 33.5% faster
Summary

Benefits

– GI at the compiler level
– Identify and explain patterns
– Apply patterns in GI
– Improves population quality and diversity

Drawbacks

– Large search spaces
– Mutation and Crossover costly
– Run-time performance measurement costs
Outlook

- Improve Amaru
  - Ease of use
  - Additional algorithms
  - Automation of pattern mining
Outlook

- Improve Amaru
  - Ease of use
  - Additional algorithms
  - Automation of pattern mining
- Additional Connectors
  - More Truffle languages
  - Additional compilers
Outlook

- Improve Amaru
  - Ease of use
  - Additional algorithms
  - Automation of pattern mining
- Additional Connectors
  - More Truffle languages
  - Additional compilers
- Answer your questions
Contact

Code available under the MIT License at https://amaru.dev

Oliver Krauss
oliver.krauss@fh-hagenberg.at

+43 (0)50804-27195
Measuring Runtime Performance

- Semantic validity → test based + coverage metrics
Measuring Runtime Performance

- Semantic validity → test based + coverage metrics
- Accurate measures for NFP → 200,000 runs per AST in own JVM
Measuring Runtime Performance

- Semantic validity → test based + coverage metrics
- Accurate measures for NFP → 200,000 runs per AST in own JVM
- Takes time

Performance over executions

Run time per execution (ns)

Execution (repetition)
Generalizable Optimizations

- Switch of variable type
- Affects example language MiniC
- Possibly generalizable
Generalizable Optimizations

- Other performance (anti)-patterns useful for GI
- Patterns can hint at issues in language
  - Ex. Inlining pattern
  - Inlining identified as performance pattern
  - Graal inlines by itself
  - Root cause was bug