Source-to-source Compilers and Meta-Programming

Advanced Software Construction Techniques
December 2022

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

DEI
DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA

Outline

> Source-to-Source Motivation

> LARA Framework

> Research

> Tutorial

SPeCS: Special-Purpose Computing Systems, Languages and Tools

> Objectives

- Systematically address non-functional requirements (e.g., execution time, energy)
- Research integrated hardware-software solutions

> Architectures

- Embedded (ARM CPUs, FPGAs)
- HPC (Intel Xeon, Xeon Phi, GPUs, FPGAs)

> Tools

- Source-to-source compilers, e.g. Clava (C/C++), Kadabra (Java), MATISSE (MATLAB)
- Binary translation framework

Compiler

> Translates code written in one language to another one

> Programming language to machine code/executable
  - gcc, clang, etc.

> Optimizing compiler
  - Not only translates, but also transforms the code

Compiler Research

> Traditional compiler tools

> Established and mature approach

> Low-level IRs
  - GIMPL, LLVM-IR

> Catalogue of existing transformations

Compiler Research - Challenges

> Some information might be lost (e.g. comments, high-level structures, loops)

> High-learning curve

> Impractical distribution

> Keep up with new compiler versions

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Source-to-Source Compiler

> Translates a high-level programming language to another high-level programming language
  - Sometimes, the same language! (e.g., C to C)

> Useful for applying automatic code transformations

> Meta-programming: programs that manipulate programs
  - Reflection
  - Compilation passes

> Commonly used in certain application areas
  - JavaScript transpilers (e.g. TypeScript)

Source-to-Source Compiler - Example

```
int foo()
{
   int a = 0;


   a+= bar1(10);


   a+= bar2(20);


   return a;
}
```

Language: C

Source-to-Source Compiler - Example

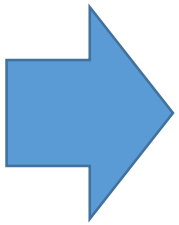
```c
int foo()
{
   int a = 0;


   a+= bar1(10);


   a+= bar2(20);


   return a;
}
```

Language: C



```c
int foo()
{
   int a = 0;


   a+= bar1(10);
   printf("foo->bar1\n");

   a+= bar2(20);
   printf("foo->bar2\n");

   return a;
}
```

Language: C

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

> High-learning curve

> Impractical distribution

> Keep up with new compiler versions

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

> Impractical distribution

> Keep up with new compiler versions

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

DSLs/APIs over AST or similar structures

> Impractical distribution

> Keep up with new compiler versions

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

DSLs/APIs over AST or similar structures

> Impractical distribution

Another tool in the tool-chain

> Keep up with new compiler versions

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

DSLs/APIs over AST or similar structures

> Impractical distribution

Another tool in the tool-chain

> Keep up with new compiler versions

Source-code as the interface

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Source-to-Source Compiler - Motivation

> Some information might be lost (e.g. comments, high-level structures, loops)

Keep all source code information

> High-learning curve

DSLs/APIs over AST or similar structures

> Impractical distribution

Another tool in the tool-chain

> Keep up with new compiler versions

Source-code as the interface

> Compiler lock-in (aggravated in fragmented environments, e.g. embedded)

Compatible with any compiler that accepts the language

LARA Framework

> Developed in the SPeCS Lab

> Source-to-source compilation
  - Code analysis and transformation using scripts (JS)

> Used in several works developed in the lab

LARA Framework Origins – Project REFLECT (2010)

Karlsruher Institut für Technologie

TU Delft

coreworks®

ACE

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

inesc id lisboa

Imperial College London

Honeywell

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

U.PORTO

LARA Framework Origins – MIEIC Dissertation (2011)

> *A meta-language and framework for aspect-oriented programming*

- Tiago Carvalho (TDRC)
- Advisor: prof. João Cardoso

> Project AMADEUS (FCT)

- *Aspects e Optimizações de Compiladores para o Desenvolvimento de sistemas com MATLAB*

LARA Framework Origins – MIEIC Dissertation (2011)

> Starts as an aspect-oriented language (AOP)

- Improve modularity by separation of concerns
- Parts of the program are “woven” into a whole
- Original AOP formulation has severe shortcomings

> LARA focused on non-functional requirements

- Instrumentation, optimization, exploration
- Similar for both C and MATLAB (target languages of the projects)

LARA Framework Origins – MIEIC Dissertation (2011)

> LARA 1.0 was relatively small and contained

- Select points in the code
- Insert code/apply pre-defined actions

Meta-Aspect code

```
aspectdef monitoringFuntionCalls
   allFunctionCalls: select function{*}.call{*} end
   apply to allFunctionCalls
        insert before %{printf("call to <$function.name>\n");}%;
   end
end
```

Figure 6: Example of a meta-aspect concerning all function calls.

LARA Framework Origins – MIEIC Dissertation (2011)

Points to select (e.g. function, call)

> LARA 1.0 was relatively small and contained

- Select points in the code
- Insert code/apply pre-defined actions

Meta-Aspect code

```
aspectdef monitoringFuntionCalls
   allFunctionCalls: select function{*}.call{*} end
   apply to allFunctionCalls
        insert before %{printf("call to <$function.name>\n");}%;
   end
end
```

Figure 6: Example of a meta-aspect concerning all function calls.

LARA Framework Origins – MIEIC Dissertation (2011)

> LARA 1.0 was relatively small and contained

- Select points in the code
- Insert code/apply pre-defined actions

Points to select (e.g. function, call)

Attributes (e.g. function.name)

Meta-Aspect code

```
aspectdef monitoringFuntionCalls
   allFunctionCalls: select function{*}.call{*} end
   apply to allFunctionCalls
        insert before %{printf("call to <$function.name>\n");}%;
   end
end
```

Figure 6: Example of a meta-aspect concerning all function calls.

LARA Framework Origins – MIEIC Dissertation (2011)

> LARA 1.0 was relatively small and contained

- Select points in the code
- Insert code/apply pre-defined actions

Points to select (e.g. function, call)

Attributes (e.g. function.name)

Actions (e.g. insert)

Meta-Aspect code

```
aspectdef monitoringFuntionCalls
   allFunctionCalls: select function{*}.call{*} end
   apply to allFunctionCalls
      insert before %{printf("call to <$function.name>\n");}%;
   end
end
```

Figure 6: Example of a meta-aspect concerning all function calls.

Codifying Strategies with LARA

strategies for optimising dataflow descriptions

arithmetic expression mapping

Hi Gabriel,

- **You can adjust DSP usage with optimization.pushDSPFactor**. - You would not expect to be DSP limited when doing floating point calculations. They burn lots of LUTs. You can force it to use DSPs for adders as well by increasing the DSP factor.
- You have a lot of dividers. These are really expensive hardware. Have a think about whether its **possible to rewrite the maths to do multiplies instead i.e. take a reciprocal of something once and then use it in many multiplies**. Multiplies are least expensive op in floating point (since they use DSP blocks which you have plenty of).
- You have two spatial loops over N in the main constructor. Assuming N!=1 this is what is causing the high resource use since you are doing the entire calculation in parallel. How big is a typical value of N? If you are streaming this data from the CPU, best case is ~2GB/s of input so for 64-bit numbers, you can only receive about 1-2 values per cycle at 150MHz. If your kernel is reading/writing more than that it will be stalling on I/O. Inst… your kernel to read M from N. e.g. If N=16, read M=4, and it will take 4 cycles to read. You need to add control to then compute… make up the full N over the course of 4 cycles. Then **by making sure M is a parameter as well as N you can adjust M to adjust the… and trade-off performance and area**. This is basically redefining what "a pipe" is. You want to… unit of co… enough to fit on the chip, then duplicate it to fill the chip up.
- **Fixed point could help a little**, but could be complex. **Cutting to 17-bit mantissa** could help. D… sive, but cost is proportional to mantissa bits.

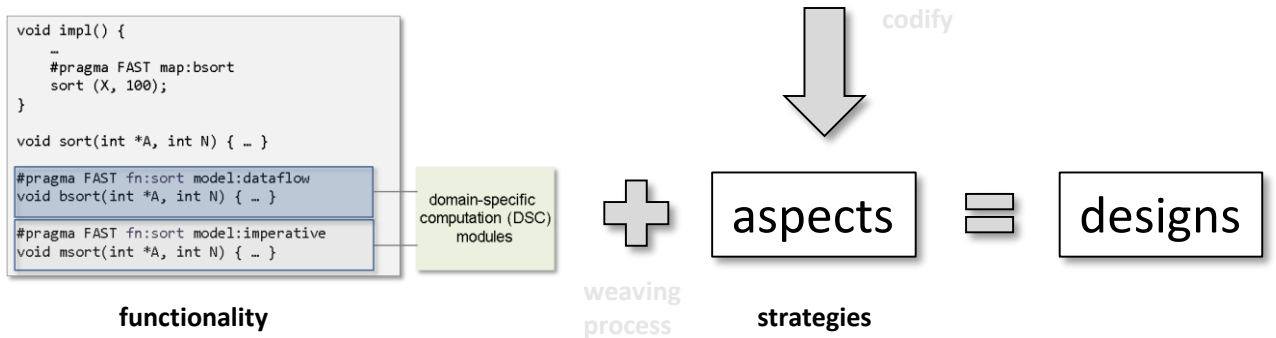
Hope this helps,

Oliver.

arithmetic expression rewriting

design pattern for trading parallelism/sharing

optimising numerical representation



LARA Framework 2.0 and Current Status

> LARA 2.0

- Adds scripting capabilities (EcmaScript 5 (2009) + LARA DSL)
- Supporting environment (IDE, documentation, testing)
- More LARA compilers

> Current status

- Supports JS-only development
- Improves multi-language support
- Focus on ergonomics and ease of use/integration

Source-to-Source Compiler - Example

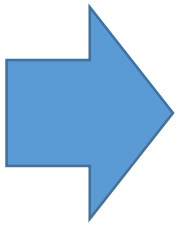
```c
int foo()
{
   int a = 0;


   a+= bar1(10);


   a+= bar2(20);


   return a;
}
```

Language: C



```c
int foo()
{
   int a = 0;


   a+= bar1(10);
   printf("foo->bar1\n");

   a+= bar2(20);
   printf("foo->bar2\n");

   return a;
}
```

Language: C

LARA Example

```
laraImport("weaver.Query");
laraImport("lara.code.Logger");

const logger = new Logger();

for(const $function of Query.search("function", "foo")) {
   for(const $call of Query.searchFrom($function, "call")) {
      logger.text($function.name + "->" + $call.name)
         .ln()
         .log($call);
   }
}
```

Language: JS

LARA Source-to-Source Compilers

> MATISSE: MATLAB-to-C/OpenCL compiler
  - specs.fe.up.pt/tools/matisse

> CLAVA: C/C++ source-to-source compiler
  - specs.fe.up.pt/tools/clava/

> KADABRA: JAVA source-to-source compiler
  - specs.fe.up.pt/tools/kadabra

> Jackdaw: JavaScript source-to-source compiler
  - specs.fe.up.pt/tools/jackdaw

> **All tools have online demos**

Research

Multi-core Processors

> Common-place now

> Require parallelization to take advantage of
  - Parallelized libraries, threads, OpenMP pragmas, etc

> Manual parallelization is not trivial
  - Can make code slower!

Auto-parallelization of C code with OpenMP (Clava)

> AutoPar: LARA library for auto parallelization of C code

- Statically analyses and inserts OpenMP pragmas
- Automatic, no user effort
- Developed by Hamid Arabnejad, SPeCS Lab post-doc researcher

```
for(int i = 0; i < numIter; i++) {
   a += i;
}
```

Language: C

Auto-parallelization of C code with OpenMP (Clava)

> AutoPar: LARA library for auto parallelization of C code

- Statically analyses and inserts OpenMP pragmas
- Automatic, no user effort
- Developed by Hamid Arabnejad, SPeCS Lab post-doc researcher

```
#pragma omp parallel for default(shared) firstprivate(numIter) reduction(+ : a)
for(int i = 0; i < numIter; i++) {
   a += i;
}
```

Language: C

Auto-parallelization of C code with OpenMP (Clava)

> Experiments:

- NAS – 2×
- PolyBench – 8.6× (8 threads, XL)
- Himeno – 10× (16 threads)

```
#pragma omp parallel for default(shared) firstprivate(numIter) reduction(+ : a)
for(int i = 0; i < numIter; i++) {
   a += i;
}
```

Language: C

Memoization (Clava)

> Result of pure functions depend only on inputs
  - sin, cos, exp
  - custom user functions
> Map lookup instead of calling function
  - Precomputed tables
  - Calculate values during runtime

> API developed by
  - Loïc Besnard, INRIA researcher of the ANTAREX project
  - Pedro Pinto, PhD student at SPeCS group

Memoization (Clava)

```
1 float foo (float p)
2 {
3   /* code of foo without side effects */
4 }
5
6 float foo_wrapper(float p)
7 {
8   float r;
9
10  /* already in the table ? */
11  if (lookup_table(p, &r)) return r;
12
13  /* calling the original function */
14  r = foo(p);
15
16  /* updating the table or not */
17  update_table(p, r);
18
19  return r;
20 }
```

Memoization (Clava)

Benchmark	Best Improvement
atmi	46%
equake	6%
fft	13%
rgb2hsi	18%

Mutation Testing (Kadabra)

> Injects common defects on the software (mutations)
  - Checks if tests pass after mutations

> However…
  - Large overhead (possibly hundreds or thousands of generated versions)
  - Not practical to test on certain systems (e.g. Android)

> Meta-mutants
  - Generate one version with all mutations
  - Requires fine-grained control of the source code

> MESW MSc theses
  - *Cost Reduction Technique for Mutation Testing* - Francisco Azevedo
  - *Mutation Operators for Android Apps* - David Mata (*ongoing*)
  - *Meta-Mutation Operators for Android Apps* - Ana Veiga (*ongoing*)

Mutation Testing (Kadabra)

```
void move(int x, int y) {
    if (java.lang.System.getProperty("MUID").equals("com.mutation.testcase_20_0")) {
        setXPos(getYPos() - x);
    }
    if (java.lang.System.getProperty("MUID").equals("com.mutation.testcase_20_1")) {
        setXPos(getYPos() * x);
    }
    if (java.lang.System.getProperty("MUID").equals("com.mutation.testcase_20_2")) {
        setXPos(getYPos() / x);
    }
    if (java.lang.System.getProperty("MUID").equals("com.mutation.testcase_20_3")) {
        setXPos(getYPos() % x);
    }
    if (java.lang.System.getProperty("MUID").equals(null)){
        setXPos(getYPos() + x);
    }
}
```

Multi-Language Support (Clava, Kadabra)

> Write LARA scripts that work on more than one language

> Already supported at API level
  - Abstract classes where language-specific functions are overriden
  - Logger, Timer, Energy

> More generic approach

Multi-Language Support (Clava, Kadabra)

> MSc theses

- *Multi-Language Software Metrics* – Gil Teixeira (MIEIC)
- *Language-Independent Detection of Design Pattern Instances* – Hugo Andrade *(MESW)*

> LARA compilers can now share a language specification

- Points, attributes, actions
- Enabled with an import
- LARA scripts fully-compatible between compilers that implement specification

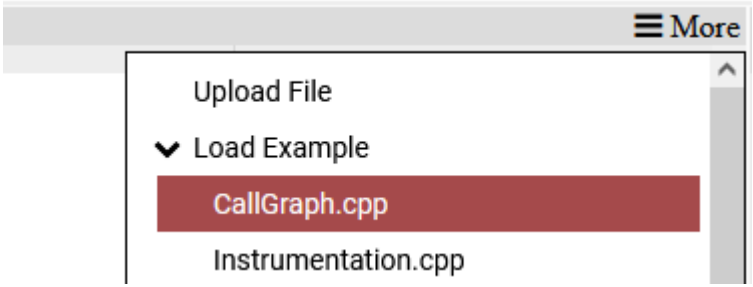
TUTORIAL

Tutorial - Preparation

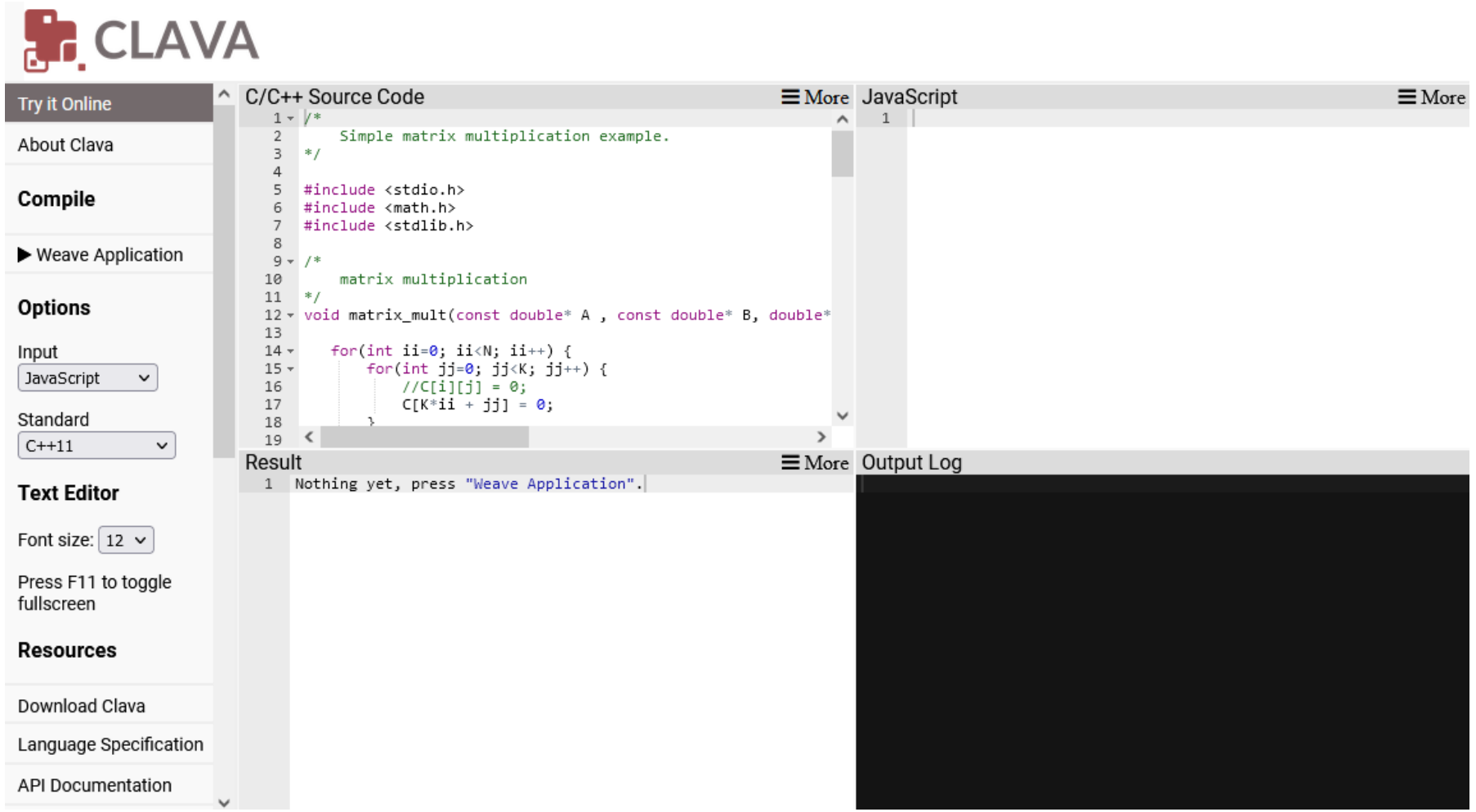
- Open Clava temporary demo website
  - **http://specs.fe.up.pt/tools/clava/**

- Delete contents of top-right area

- To reload C/C++ example:
  - More->Load Example->CallGraph.cpp



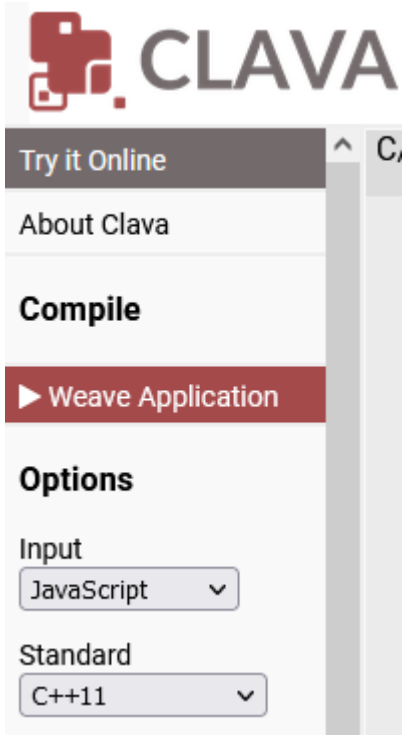
Tutorial - Preparation



Tutorial – Hello World

- JavaScript box
  - `println("Hello World")`

- Press “Weave Application”



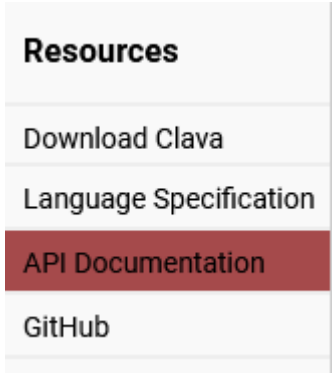
Tutorial – Node types (Ex1)

Use Query API to get and print the AST root node

Tutorial – Node types (Ex1)

Use Query API to get and print the AST root node

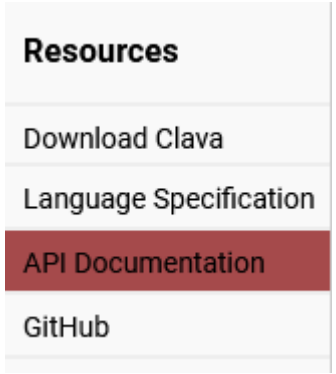
- Check API Documentation



Tutorial – Node types (Ex1)

Use Query API to get and print the AST root node

- Check API Documentation



- Find “Query” class

- Find out how to get the root node

Tutorial – Node types (Ex1)

Use Query API to get and print the AST root node

```
laraImport("weaver.Query");

println(Query.root());
```

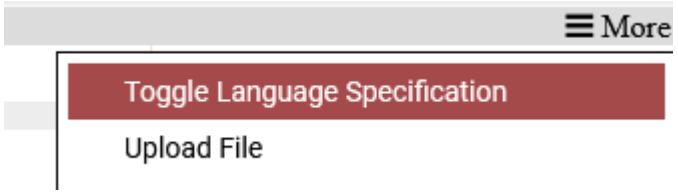
Tutorial – Node Attributes (Ex2)

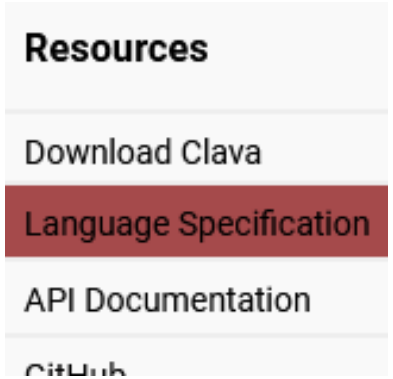
Print code of root node

Tutorial – Node Attributes (Ex2)

Print code of root node

- Language Specification
  - Toggle Language Specification
  - …or open Language Specification page
- Find attribute ‘code’
- To access attribute code: `node.code`



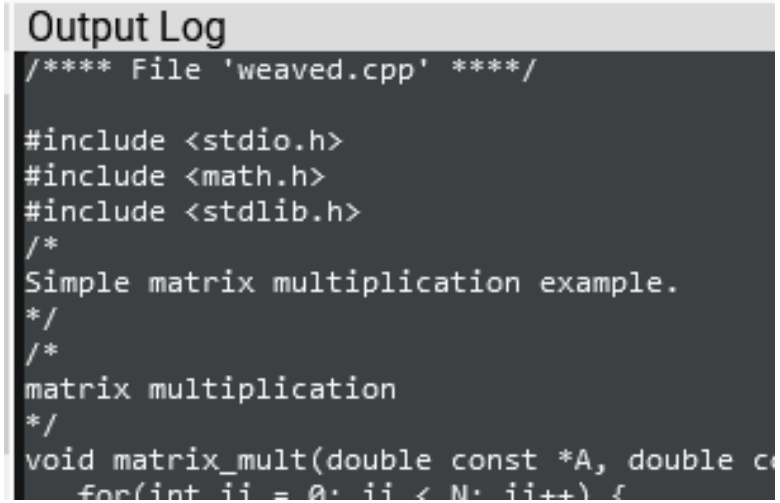


Tutorial – Node Attributes (Ex2)

Print code of root node

```
laraImport("weaver.Query");

println(Query.root().code);
```



Tutorial – Node Attributes (Ex3)

Print filenames of the files of the program

Tutorial – Node Attributes (Ex3)

Print filenames of the files of the program

- Get files from an attribute in program

- Get filename from an attribute in file

Tutorial – Node Attributes (Ex3)

Print filenames of the files of the program

```
laraImport("weaver.Query");

println(Query.root().files
   .map(file => file.filename))
```

Tutorial – AST Structure (Ex4)

What are the types of the children of a ‘function’ node?

Tutorial – AST Structure (Ex4)

What are the types of the children of a 'function' node?

- Use the attribute 'dump' to print the AST

Tutorial – AST Structure (Ex4)

What are the types of the children of a 'function' node?

R.: param and body

```
laraImport("weaver.Query");

println(Query.root().dump);
```

Output Log

```
'program'
  'file'
    'include'
    'include'
    'include'
    'comment'
    'comment'
    'function'
      'param'
      'param'
      'param'
      'param'
      'param'
      'param'
      'body'
        'loop'
          'declStmt'
            'vardecl'
              'intLiteral'
          'exprStmt'
            'binaryOp'
              'varref'
              'varref'
```

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter ‘m’

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter ‘m’

- Check documentation of “Query” class
- Check the method “search()” of Query

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter 'm'

R.: matrix_mult, main


```
laraImport("weaver.Query");

println(Query.search("function", {name: /^m.*/})
             .get()
             .map(f => f.name))
```

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter 'm'

R.: matrix_mult, main

Output Log
```
matrix_mult
main
```

```
laraImport("weaver.Query");

for(const f of Query.search("function", {name: /^m.*/}))
{
    println(f.name)
}
```

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter 'm'

R.: matrix_mult, main

Output Log
```
matrix_mult
main
```

```
laraImport("weaver.Query");

for(const f of Query.search("function",
                            {name: n => n.startsWith("m")})) {
    println(f.name)
}
```

Tutorial – AST Navigation (Ex6)

Print all <function> - <call> pairs

Tutorial – AST Navigation (Ex6)

Print all <function> - <call> pairs

- E.g. `main->test_matrix_mul, test_matrix_mul->matrix_mult,…`
- `Query.search()` returns a `weaver.Selector`
- Chained searches
  - Method 1: `Selector.search()` + `Selector.chain()`
  - Method 2: Nested `for` + `Query.searchFrom()`

Tutorial – AST Navigation (Ex6)

Print all <function> - <call> pairs

Output Log

```
init_matrix->rand,print_matrix_result->printf,
```

```
laraImport("weaver.Query");


println(Query.search("function")
             .search("call")
             .chain()
             .map(ch => ch.function.name
                        +"->"+ch.call.name)
)
```

Tutorial – AST Navigation (Ex6)

Print all <function> - <call> pairs

Output Log

```
init_matrix->rand
print_matrix_result->printf
test_matrix_mul->malloc
```

```
laraImport("weaver.Query");

for(const f of Query.search("function")) {
 for(const c of Query.searchFrom(f, "call")) {
   println(f.name+"->"+c.name)
 }
}
```

Tutorial – AST Transformation (Ex7)

Insert a comment before each call, with format “// <call_name>”

Tutorial – AST Transformation (Ex7)

Insert a comment before each call, with format "// <call_name>"

- "Actions" section in Language Specification
- Actions change the AST
- Actions for inserting code
  - `insertBefore(node|String)`
  - `insertAfter(node|String)`

Tutorial – AST Transformation (Ex7)

Insert a comment before each call, with format "// <call_name>"

```
laraImport("weaver.Query");

Query.search("call").get()
     .forEach(c => c.insertBefore("// "+c.name))
```

```
int main() {
   // To make results repeatable
   // srand
   srand(0);
   // test_matrix_mul
   test_matrix_mul();
}
```

Tutorial – AST Transformation (Ex8)

Insert a printf before each call that prints “<call_name>@<line>”

Tutorial – AST Transformation (Ex8)

Insert a printf before each call that prints "<call_name>@<line>"

- JavaScript template literals/string might help

- Can use `Clava.rebuild()` to test generated code syntax
  - `laraImport("clava.Clava")`

Tutorial – AST Transformation (Ex8)

Insert a printf before each call that prints "<call_name>@<line>"

```
laraImport("weaver.Query");


Query.search("call")
      .get()
      .forEach(c => c.insertBefore(
         `printf("${c.name}@${c.line}\\n");`
        )
      )
```

```
int main() {
   printf("srand@86\n");
   // To make results repeatable
   srand(0);
   printf("test_matrix_mul@88\n");
   test_matrix_mul();
}
```

Tutorial – AST Transformation

- What about includes?

Tutorial – AST Transformation

- What about includes?
  - `call.ancestor("file").addInclude("stdio.h", true)`

Tutorial – AST Transformation

- What about includes?
  - `call.ancestor("file").addInclude("stdio.h", true)`

- What if C++?

Tutorial – AST Transformation

- What about includes?
  - `call.ancestor("file").addInclude("stdio.h", true)`

- What if C++?
  - `Clava.isCxx()`

Tutorial – AST Transformation

- What about includes?
  - `call.ancestor("file").addInclude("stdio.h", true)`
- What if C++?
  - `Clava.isCxx()`
- Seems a lot of work, is there a better way?

Tutorial – AST Transformation

- What about includes?
  - `call.ancestor("file").addInclude("stdio.h", true)`
- What if C++?
  - `Clava.isCxx()`
- Seems a lot of work, is there a better way?
  - Encapsulate complex functionality in an API

Tutorial – AST Transformation (Ex9)

Insert a printf before each call that prints “<call_name>@<line>”, using the Logger API (i.e. lara.code.Logger)

Tutorial – AST Transformation (Ex9)

Insert a printf before each call that prints "<call_name>@<line>", using the Logger API (i.e. lara.code.Logger)

- Instantiate a Logger object

- Methods `text()`, `ln()` and `log()` might be useful

Tutorial – AST Transformation (Ex9)

Insert a printf before each call that prints "<call_name>@<line>", using the Logger API (i.e. lara.code.Logger)

```
laraImport("weaver.Query");
laraImport("lara.code.Logger");
const logger = new Logger();

Query.search("call").get().forEach(c =>
        logger.text(`${c.name}@${c.line}`)
        .ln()
        .log(c, true))
```

```
int main() {
   std::cout << "srand@86" << "\n";
   // To make results repeatable
   srand(0);
   std::cout << "test_matrix_mul@88" << "\n";
   test_matrix_mul();
}
```

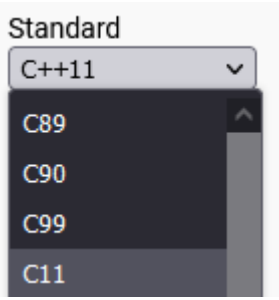
Tutorial – AST Transformation (Ex9)

Insert a printf before each call that prints "<call_name>@<line>", using the Logger API (i.e. lara.code.Logger)

```
laraImport("weaver.Query");
laraImport("lara.code.Logger");
const logger = new Logger();

Query.search("call").get().forEach(c =>
        logger.text(`${c.name}@${c.line}`)
        .ln()
        .log(c, true))
```

```
int main() {
   printf("srand@86\n");
   // To make results repeatable
   srand(0);
   printf("test_matrix_mul@88\n");
   test_matrix_mul();
}
```



Tutorial – AST Transformation (Ex10)

Use Timer API to measure execution time of `matrix_mult` calls

Tutorial – AST Transformation (Ex10)

Use Timer API to measure execution time of `matrix_mult` calls

- Import `lara.code.Timer`

Tutorial – AST Transformation (Ex10)

Use Timer API to measure execution time of `matrix_mult` calls

```
laraImport("weaver.Query");
laraImport("lara.code.Timer");
const timer = new Timer();
```

```
std::chrono::high_resolution_clock::
// do: C = A*B
matrix_mult(A, B, C, N, M, K);
std::chrono::high_resolution_clock::
auto clava_timing_duration_0 = std::
std::cout << "Time matrix_mult@77:"
```

```
Query.search("call", "matrix_mult")
      .get()
      .forEach(c =>
            timer.time(c,`Time ${c.name}@${c.line}:`))
```

The End

THANKS!

CMake Package

- Apply Clava scripts to CMake C/C++ projects

- Add Clava CMake package
  - https://github.com/specs-feup/clava/tree/master/CMake

Object-Oriented Common Language Specification