MetaFork Documentation

Compute Science of UWO
Xiaohui Chen
Marc Moreno Maza

June 10, 2014

1 MetaFork Usage

1.1 Requirements

- Linux systems, i.e. Ubuntu, using bash.
- GCC version 4.8.* or later to support CilkPlus and OpenMP.

1.2 Installation

One should set these variables in the ".bashrc" file for installing and running related libraries.

- export PATH=your gcc path/location/bin:$PATH
- export LIBRARY_PATH=your gcc path/location/lib:$LIBRARY_PATH
- export LD_LIBRARY_PATH=your gcc path/location/lib:$LD_LIBRARY_PATH
- export METAFORKHOME=your path where you save metafork directory
- export PATH=$METAFORKHOME/src/bin:$PATH

Run "source .bashrc" once it is done.

Download the software from our website: www.metafork.org/downloads.html

- tar -zxvf metafork_distrib.tgz
- cd metafork
- make translators

Note: "make check" This target can be used to verify that the Metafork translators work correctly with the examples.

1.3 How to Run

Supposing you have already downloaded and compiled our source code, and have the binary codes of our translators.

One can put any number of files in the command line and also directory translation is supported.

For example:

metatoopenmp a.meta ../../b.meta

the new generated file a.openmp and ../../b.openmp will be put in the same directory of a.meta and ../../b.meta

metatoopenmp src/ dest/

if dest/ directory does not exist, then we will translate all files from src/ directory to dest/.

if dest/ directory exists, then we will translate all the files in src/ and dest/ directory locally.

metatoopenmp src/ -d dest/

"-d" indicates that dest/ is the target directory. In any case we will translate all the files from src/ to dest/.

Also you can print some debug information using "-v", "-verbose" or "–verbose".

In the command line only one "-d" is permitted. For example: the following cases is allowed:

metatoopenmp a.meta src/ -d dest/
metatoopenmp a.meta src/ -v -d dest/
metatoopenmp a.meta src/ -d dest/ -v

the following is not allowed:

metatoopenmp a.meta -d dest/ src/
metatoopenmp a.meta src/ -d -v dest/
metatoopenmp a.meta src/ -d dest/ -d dest2/

2 AST

2.1 General problem

When we implemented our translators, we have to use macros and ifedfs, so we looked into several papers to understand this question and that follows is a review of this literature.

A programming language is incomplete without the C pre-processor, Cpp [1]. As the designer and original implementor of C++, Bjarne Stroustrup notes that "without the C preprocessor, C itself ... would have been stillborn" [2].

Programmers can use Cpp to define constants, define new syntax, abbreviate repetitive or complicated constructs, support conditional compilation, and reduce or eliminate reliance on a compiler implementation.

Using these smart tricks, it can reduce programmer effort and improve portability and readability.

While from another point of view, Cpp is widely viewed as a source of difficulty for understanding and modifying C programs because of these tricks.

For example, when in the source file we encounter a pre-token "A", it is not necessarily the case that "A" is an identifier, since it maybe a marco which will be expanded later.

For "ifdefs", it is more complicated which may change the flow of the code.

So it is quite a challenge to develop style-preserving source-to-source transformation tools for C and C++. The main difficulty does not come from the

complexity of those languages, but the general use of the Cpp, especially ifdefs and macros.

Developing source-to-source transformation tools is much easier as original source code is pre-processed so macros and ifdefs disappear.

Some of the source-to-source translators use this method like OMPI [3].

The disadvantage is obvious. For example the standard headfile "iostream" will be expanded after pre-processing. This makes the pre-processed code is quite big and unreadable since "iostream " has about 17427 lines.

In paper [4], the authors survey a great deal of publicly available C code to determine how Cpp is used in practice, and the results show that most Cpp usage follows fairly simple patterns.

According to the above conclusion, in paper [5], the authors normalize the macros and ifdefs according to heuristics and add them to the grammar as an ordinary token to parse C/C++ source files without preprocessing. In the paper it says that their method still can not parse most of the C++ code in the C++ standard GNU library as this code uses advanced C++ features and macros not yet handled by their heuristics and grammar.

2.2 Our solution

Our goal is to preserve the original code format as much as possible, so we need to handle these Cpps in our AST. This part is still under debugging.

For Macros, when parser encounters a token, first it checks whether it is a macro, if yes, the macro will be expanded in AST using yy_scan_string, also in AST we will mark that this is a macro, so when outputting the AST we just output the macro but not the content of that macro.

For ifdefs, all ifdefs will be evaluated. After evaluation, only one possible configuration of the source is processed like normal procedure. For the other configuration which is not satisfied, we do not parse the contents of that part, we instead record it in AST verbatim like comments and also output verbatim.

One exception is that for the ifdefs inside a parallel region, sometimes the parallel region will be outlined, so it is possible that the scope of the ifdefs will change and the sequence is that the behavior of the code will be affected. In this case, our solution is that the unsatisfied part will be discarded and we just parse the eligible branch, so when outputting the AST we will lose parts of the code. The programmer should avoid using ifdefs inside a parallel region.

Another problem that one should notice is that users can pass Macros through the command line. For example in gcc and g++, one can use option -D, it is not recommended, since this information can not be caught by our translators.

3 Translation examples

Here are some examples showing how to translate codes among CILKPLUS, OPENMP and METAFORK.

```
long fib(long n)
{
    long x, y;
    if (n<2) return n;
    else if (n<BASE)
        return fib_serial(n);
    else
    {
        x = cilk_spawn fib(n-1);
        y = fib(n-2);
        cilk_sync;
        return (x+y);
    }
}
```

```
long fib(long n)
{
    long x, y;
    if (n<2) return n;
    else if (n<BASE)
        return fib_serial(n);
    else
    {
        x = meta_fork fib(n-1);
        y = fib(n-2);
        meta_join;
        return (x+y);
    }
}
```

Figure 1: Example of translating function spawn from CILKPLUS to METAFORK.

```
long fib(long n)
{
    long x, y;
    if (n<2) return n;
    else if (n<BASE)
        return fib_serial(n);
    else
    {
        x = meta_fork fib(n-1);
        y = fib(n-2);
        meta_join;
        return (x+y);
    }
}
```

```
long fib(long n)
{
    long x, y;
    if (n<2) return n;
    else if (n<BASE)
        return fib_serial(n);
    else
    {
        #pragma omp task shared(x)
        x = fib(n-1);
        y = fib(n-2);
        #pragma omp taskwait
        return (x+y);
    }
}
```

Figure 2: Example of translating function spawn from METAFORK to OPENMP.

```
int main()
{
  int a[ N ];
  int b = 0;
  #pragma omp parallel
  #pragma omp for private(b)
  for(int i = 0; i < N; i++)
  {
    b = i ;
    a[ i ] = b;
  }
}
```

```
int main()
{
  int a[ N ];
  int b = 0;

  meta_for (int i = 0; i < N; i++)
  {
    int b;
    b = i ;
    a[ i ] = b;
  }
}
```

Figure 3: Example of translating parallel region from OpenMP to MetaFork.

```
int main()
{
  int sum_a=0, sum_b=0;
  int a[ 5 ] = {0,1,2,3,4};
  int b[ 5 ]= {0,1,2,3,4};

  meta_fork shared(sum_a){
      for(int i=0; i<5; i++)
        sum_a += a[ i ];
      }

  meta_fork shared(sum_b){
      for(int i=0; i<5; i++)
         sum_b += b[ i ];
      }

  meta_join;
}
```

```
void fork_func0(int* sum_a, int* a)
{
        for(int i=0; i<5; i++)
          (*sum_a) += a[ i ];
}
void fork_func1(int* sum_b, int* b)
{
        for(int i=0; i<5; i++)
           (*sum_b) += b[ i ];
}
int main()
{
  int sum_a=0, sum_b=0;
  int a[ 5 ] = {0,1,2,3,4};
  int b[ 5 ] = {0,1,2,3,4};
  cilk_spawn fork_func0(&sum_a, a);
  cilk_spawn fork_func1(&sum_b, b);
  cilk_sync;
}
```

Figure 4: Example of translating parallel region from MetaFork to Cilkplus.

```
void main()
{
   int i, j;
   #pragma omp parallel
   {
     #pragma omp sections
      {
         #pragma omp section
         {
             i++;
         }
         #pragma omp section
         {
             j++;
         }
      }
   }
}
```

```
void main()
{
  int i, j;
  {
    meta_fork shared(i)
    {
        i++;
    }
    meta_fork shared(j)
    {
        j++;
    }
    meta_join;
  }
}
```

Figure 5: Example of translating parallel region (sections) from OpenMP to MetaFork.

4 Grammar

In our translators, we add some our own grammars based on the ISO standard C/C++ grammar [6] without changing any of the original grammars.

In the appendix part, the syntactic expression $term_{opt}$ indicates that the term is optional.

A MetaFork grammar

jump-statement:
/*original rules plus following rule*/
| METAJOIN ';'
;
statement:
/*original rules plus following rule*/
| meta_construct
;
structured_block:
statement
;
meta_construct:
METAFORK SHARED$_{opt}$ '($_{opt}$' variable_list$_{opt}$ ')$_{opt}$' structured_block
;
grainsize-pragma:
pragma meta grainsize = expression new-line
;
in C:
iteration-statement:
/*original rules plus following rule*/
| grainsize-pragmaopt METAFOR (declaration expressionopt ; expressionopt) statement
;
in cplusplus:
iteration-statement:
/*original rules plus following rule*/
| grainsize-pragmaopt METAFOR (for-init-statement conditionopt ; expressionopt) statement
;

B Openmp grammar

declaration:
/*original rules plus following two rules*/
| threadprivate_directive
;
statement:
/*original rules plus following two rules*/
| openmp_construct
;
in C:
block_item:
/*original rules plus following two rules*/

| openmp_directive
;
in cplusplus:
statement-seq:
/*original rules plus following two rules*/
| openmp_directive
| statement-seq openmp_directive
;
openmp_construct:
parallel_construct
| for_construct
| sections_construct
| single_construct
| parallel_for_construct
| parallel_sections_construct
| master_construct
| critical_construct
| atomic_construct
| ordered_construct
| task_construct
;
openmp_directive:
barrier_directive
| flush_directive
| taskwait_directive
| taskyield_directive
;
structured_block:
statement
;
parallel_construct:
parallel_directive structured_block
;
parallel_directive:
PRAGMA_OMP OMP_PARALLEL parallel_clause_optseq NEWLINE
;
parallel_clause_optseq:
// empty
| parallel_clause_optseq parallel_clause
| parallel_clause_optseq ',' parallel_clause
;
parallel_clause:
unique_parallel_clause
| data_clause
;
unique_parallel_clause:

```
    OMP_IF '(' sc_pause_openmp(); expression ')'
    | OMP_NUMTHREADS '(' sc_pause_openmp(); expression ')'
    ;
for_construct:
    for_directive iteration_statement_for
    ;
for_directive:
    PRAGMA_OMP OMP_FOR for_clause_optseq NEWLINE
    ;
for_clause_optseq:
    // empty
    | for_clause_optseq for_clause
    | for_clause_optseq ',' for_clause
    ;
for_clause:
    unique_for_clause
    | data_clause
    | OMP_NOWAIT
    ;
unique_for_clause:
    OMP_ORDERED
    | OMP_SCHEDULE '(' schedule_kind ')'
    | OMP_SCHEDULE '(' schedule_kind ',' sc_pause_openmp(); expression ')'
    | OMP_SCHEDULE '(' OMP_AFFINITY ',' expression ')'
    | OMP_COLLAPSE '(' expression /* CONSTANT */ ')' /* OpenMP V3.0
*/
    ;
schedule_kind:
    OMP_STATIC
    | OMP_DYNAMIC
    | OMP_GUIDED
    | OMP_RUNTIME
    | OMP_AUTO
    ;
sections_construct:
    sections_directive section_scope
    ;
sections_directive:
    PRAGMA_OMP OMP_SECTIONS sections_clause_optseq NEWLINE
    ;
sections_clause_optseq:
    // empty
    | sections_clause_optseq sections_clause
    | sections_clause_optseq ',' sections_clause
    ;
sections_clause:
```

data_clause
| OMP_NOWAIT
;
section_scope:
'' section_sequence ''
;
section_sequence:
structured_block
| section_directive structured_block
| section_sequence section_directive structured_block
;
section_directive:
PRAGMA_OMP OMP_SECTION NEWLINE
;
single_construct:
single_directive structured_block
;
single_directive:
PRAGMA_OMP OMP_SINGLE single_clause_optseq NEWLINE
;
single_clause_optseq:
// empty
| single_clause_optseq single_clause
| single_clause_optseq ',' single_clause
;
single_clause:
data_clause
| OMP_NOWAIT
;
parallel_for_construct:
parallel_for_directive iteration_statement_for
;
parallel_for_directive:
PRAGMA_OMP OMP_PARALLEL OMP_FOR parallel_for_clause_optseq NEW-
LINE
;
parallel_for_clause_optseq:
// empty
| parallel_for_clause_optseq parallel_for_clause
| parallel_for_clause_optseq ',' parallel_for_clause
;
parallel_for_clause:
unique_parallel_clause
| unique_for_clause
| data_clause
;

```
parallel_sections_construct:
    parallel_sections_directive section_scope
    ;
parallel_sections_directive:
    PRAGMA_OMP OMP_PARALLEL OMP_SECTIONS parallel_sections_clause_optseq
NEWLINE
    ;
parallel_sections_clause_optseq:
    // empty
    | parallel_sections_clause_optseq parallel_sections_clause
    | parallel_sections_clause_optseq ',' parallel_sections_clause
    ;
parallel_sections_clause:
    unique_parallel_clause
    | data_clause
    ;
task_construct:
    task_directive structured_block
    ;
task_directive:
    PRAGMA_OMP OMP_TASK task_clause_optseq NEWLINE
    ;
task_clause_optseq:
    // empty
    | task_clause_optseq task_clause
    | task_clause_optseq ',' task_clause
    ;
task_clause:
    unique_task_clause
    | data_clause
    ;
unique_task_clause:
    OMP_IF '(' expression ')'
    | OMP_UNTIED
    | OMP_FINAL '(' expression ')'
    | OMP_MERGEABLE
    ;
master_construct:
    master_directive structured_block
    ;
master_directive:
    PRAGMA_OMP OMP_MASTER NEWLINE
    ;
critical_construct:
    critical_directive structured_block
    ;
```

```
critical_directive:
    PRAGMA_OMP OMP_CRITICAL NEWLINE
    | PRAGMA_OMP OMP_CRITICAL region_phrase NEWLINE
    ;
region_phrase:
    '(' IDENTIFIER ')'
    ;
taskwait_directive:
    PRAGMA_OMP OMP_TASKWAIT NEWLINE
    ;
taskyield_directive:
    PRAGMA_OMP OMP_TASKYIELD NEWLINE
    ;
barrier_directive:
    PRAGMA_OMP OMP_BARRIER NEWLINE
    ;
atomic_construct:
    atomic_directive expression_statement
    ;
atomic_directive:
    PRAGMA_OMP OMP_ATOMIC NEWLINE
    | PRAGMA_OMP OMP_ATOMIC OMP_WRITE NEWLINE
    | PRAGMA_OMP OMP_ATOMIC OMP_READ NEWLINE
    | PRAGMA_OMP OMP_ATOMIC OMP_UPDATE NEWLINE
    ;
flush_directive:
    PRAGMA_OMP OMP_FLUSH NEWLINE
    | PRAGMA_OMP OMP_FLUSH flush_vars NEWLINE
    ;
flush_vars:
    '(' variable_list ')'
    ;
ordered_construct:
    ordered_directive structured_block
    ;
ordered_directive:
    PRAGMA_OMP OMP_ORDERED NEWLINE
    ;
threadprivate_directive:
    PRAGMA_OMP_THREADPRIVATE '(' thrprv_variable_list ')' NEWLINE
    ;
data_clause:
    OMP_PRIVATE '(' variable_list ')'
    | OMP_COPYPRIVATE '(' variable_list ')'
    | OMP_FIRSTPRIVATE '(' variable_list ')'
    | OMP_LASTPRIVATE '(' variable_list ')'
```

| OMP_SHARED '(' variable_list ')'
| OMP_DEFAULT '(' OMP_SHARED ')'
| OMP_DEFAULT '(' OMP_NONE ')'
| OMP_REDUCTION '(' reduction_operator ':' variable_list ')'
| OMP_COPYIN '(' variable_list ')'
;
reduction_operator:
'+'
| '*'
| '-'
| '&'
| '^'
| '|'
| AND_OP
| OR_OP
| OMP_MIN
| OMP_MAX
;
variable_list:
IDENTIFIER
| variable_list ',' IDENTIFIER
;
thrprv_variable_list:
IDENTIFIER
| thrprv_variable_list ',' IDENTIFIER
;

C CilkPlus Grammar

jump-statement:
/*original rules plus following rule*/
| CILKSYNC ;
;
postfix-expression:
/*original rules plus following rule*/
| CILKSPAWN postfix-expression (expression-list$_{opt}$)
;
grainsize-pragma:
pragma cilk grainsize = expression new-line
;
in C:
iteration-statement:
/*original rules plus following rule*/
| grainsize-pragma$_{opt}$ CILKFOR (declaration expression$_{opt}$; expression$_{opt}$)
statement

;
in cplusplus:
iteration-statement:
/*original rules plus following rule*/
| grainsize-pragma$_{opt}$ CILKFOR (for-init-statement condition$_{opt}$; expression$_{opt}$) statement
;

D C++ grammar

D.1 C++ Expression

primary-expression:
literal
| this
| (expression)
| id-expression
| lambda-expression
;
id-expression:
unqualified-id
| qualified-id
;
unqualified-id:
identifier
| operator-function-id
| conversion-function-id
| literal-operator-id
| ~ class-name
| ~ decltype-specifier
| template-id
;
qualified-id:
nested-name-specifier template$_{opt}$ unqualified-id
| :: identifier
| :: operator-function-id
| :: literal-operator-id
| :: template-id
;
nested-name-specifier:
::$_{opt}$ type-name ::
| ::$_{opt}$ namespace-name ::
| decltype-specifier ::
| nested-name-specifier identifier ::
| nested-name-specifier template$_{opt}$ simple-template-id ::

;
lambda-expression:
lambda-introducer lambda-declarator$_{opt}$ compound-statement
;
lambda-introducer:
[lambda-capture$_{opt}$]
;
lambda-capture:
capture-default
| capture-list
| capture-default , capture-list
;
capture-default:
&
| =
;
capture-list:
capture ...$_{opt}$
| capture-list , capture ...$_{opt}$
;
capture:
identifier
| & identifier
| this
;
lambda-declarator:
(parameter-declaration-clause) mutable$_{opt}$
| exception-specification$_{opt}$ attribute-specifier-seq$_{opt}$ trailing-return-type$_{opt}$
;
postfix-expression:
primary-expression
| postfix-expression [expression]
| postfix-expression [braced-init-list]
| postfix-expression (expression-list$_{opt}$)
| simple-type-specifier (expression-list$_{opt}$)
| typename-specifier (expression-list$_{opt}$)
| simple-type-specifier braced-init-list
| typename-specifier braced-init-list
| postfix-expression . template$_{opt}$ id-expression
| postfix-expression − > template$_{opt}$ id-expression
| postfix-expression . pseudo-destructor-name
| postfix-expression − > pseudo-destructor-name
| postfix-expression ++
| postfix-expression −−
| dynamic_cast < type-id > (expression)
| static_cast < type-id > (expression)

| reinterpret_cast < type-id > (expression)
| const_cast < type-id > (expression)
| typeid (expression)
| typeid (type-id)
;

expression-list:
initializer-list
;

pseudo-destructor-name:
nested-name-specifier$_{opt}$ type-name :: type-name
| nested-name-specifier template simple-template-id :: type-name
| nested-name-specifier$_{opt}$ type-name
| decltype-specifier
;

unary-expression:
postfix-expression
| ++ cast-expression
| −− cast-expression
| unary-operator cast-expression
| sizeof unary-expression
| sizeof (type-id)
| sizeof ... (identifier)
| alignof (type-id)
| noexcept-expression
| new-expression
| delete-expression
;

unary-operator:
*
| &
| +
| -
| !
| ∼
;

new-expression:
::$_{opt}$new new-placement$_{opt}$ new-type-id new-initializer$_{opt}$
| ::$_{opt}$new new-placement$_{opt}$(type-id) new-initializer$_{opt}$
;

new-placement:
(expression-list)
;

new-type-id:
type-specifier-seq new-declarator$_{opt}$
;

new-declarator:

ptr-operator new-declarator$_{opt}$
| noptr-new-declarator
;
noptr-new-declarator:
[expression] attribute-specifier-seq$_{opt}$
| noptr-new-declarator [constant-expression] attribute-specifier-seq$_{opt}$
;
new-initializer:
(expression-list$_{opt}$)
| braced-init-list
;
delete-expression:
::$_{opt}$delete cast-expression
| ::$_{opt}$delete [ ] cast-expression
;
noexcept-expression:
noexcept (expression)
;
cast-expression:
unary-expression
| (type-id) cast-expression
;
pm-expression:
cast-expression
| pm-expression .* cast-expression
| pm-expression $->*$ cast-expression
;
multiplicative-expression:
pm-expression
| multiplicative-expression * pm-expression
| multiplicative-expression / pm-expression
| multiplicative-expression % pm-expression
;
additive-expression:
multiplicative-expression
| additive-expression + multiplicative-expression
| additive-expression - multiplicative-expression
;
shift-expression:
additive-expression
| shift-expression << additive-expression
| shift-expression >> additive-expression
;
relational-expression:
shift-expression
| relational-expression < shift-expression

```
    | relational-expression > shift-expression
    | relational-expression <= shift-expression
    | relational-expression >= shift-expression
    ;
equality-expression:
    relational-expression
    | equality-expression == relational-expression
    | equality-expression != relational-expression
    ;
and-expression:
    equality-expression
    | and-expression & equality-expression
    ;
exclusive-or-expression:
    and-expression
    | exclusive-or-expression ^ and-expression
    ;
inclusive-or-expression:
    exclusive-or-expression
    | inclusive-or-expression | exclusive-or-expression
    ;
logical-and-expression:
    inclusive-or-expression
    | logical-and-expression && inclusive-or-expression
    ;
logical-or-expression:
    logical-and-expression
    | logical-or-expression || logical-and-expression
    ;
conditional-expression:
    logical-or-expression
    | logical-or-expression ? expression : assignment-expression
    ;
assignment-expression:
    conditional-expression
    | logical-or-expression assignment-operator initializer-clause
    | throw-expression
    ;
assignment-operator:
    =
    | *=
    | /=
    | %=
    | +=
    | -=
    | >>=
```

| <<=
| &=
| ^ =
| |=
;
expression:
assignment-expression
| expression , assignment-expression
;
constant-expression:
conditional-expression
;

D.2 C++ Statement

statement:
labeled-statement
| attribute-specifier-seq$_{opt}$ expression-statement
| attribute-specifier-seq$_{opt}$ compound-statement
| attribute-specifier-seq$_{opt}$ selection-statement
| attribute-specifier-seq$_{opt}$ iteration-statement
| attribute-specifier-seq$_{opt}$ jump-statement
| declaration-statement
| attribute-specifier-seq$_{opt}$ try-block
;
labeled-statement:
attribute-specifier-seq$_{opt}$ identifier : statement
| attribute-specifier-seq$_{opt}$case constant-expression : statement
| attribute-specifier-seq$_{opt}$default : statement
;
expression-statement:
expression$_{opt}$;
;
compound-statement:
statement-seq$_{opt}$
;
statement-seq:
statement
| statement-seq statement
;
selection-statement:
if (condition) statement
| if (condition) statement else statement
| switch (condition) statement
;
condition:

expression
| attribute-specifier-seq$_{opt}$ decl-specifier-seq declarator = initializer-clause
| attribute-specifier-seq$_{opt}$ decl-specifier-seq declarator braced-init-list
;

iteration-statement:
while (condition) statement
| do statement while (expression) ;
| for (for-init-statement condition$_{opt}$; expression$_{opt}$) statement
| for (for-range-declaration : for-range-initializer) statement
;

for-init-statement:
expression-statement
| simple-declaration
;

for-range-declaration:
attribute-specifier-seq$_{opt}$ decl-specifier-seq declarator
;

for-range-initializer:
expression
| braced-init-list
;

jump-statement:
break ;
| continue ;
| return expression$_{opt}$;
| return braced-init-list ;
| goto identifier ;
;

declaration-statement:
block-declaration
;

D.3 C++ Declarations

declaration-seq:
declaration
| declaration-seq declaration
;

declaration:
block-declaration
| function-definition
| template-declaration
| explicit-instantiation
| explicit-specialization
| linkage-specification
| namespace-definition

| empty-declaration
| attribute-declaration
;

block-declaration:
simple-declaration
| asm-definition
| namespace-alias-definition
| using-declaration
| using-directive
| static_assert-declaration
| alias-declaration
| opaque-enum-declaration
;

alias-declaration:
using identifier attribute-specifier-seq$_{opt}$ = type-id ;
;

simple-declaration:
decl-specifier-seq$_{opt}$ init-declarator-list$_{opt}$;
| attribute-specifier-seq decl-specifier-seq$_{opt}$ init-declarator-list ;
;

static_assert-declaration:
static_assert (constant-expression , string-literal) ;
;

empty-declaration:
;

attribute-declaration:
attribute-specifier-seq ;
;

decl-specifier:
storage-class-specifier
| type-specifier
| function-specifier
| friend
| typedef
| constexpr
;

decl-specifier-seq:
decl-specifier attribute-specifier-seq$_{opt}$
| decl-specifier decl-specifier-seq
;

storage-class-specifier:
register
| static
| thread_local
| extern
| mutable

;

function-specifier:
inline
| virtual
| explicit
;

typedef-name:
identifier
;

type-specifier:
trailing-type-specifier
| class-specifier
| enum-specifier
;

trailing-type-specifier:
simple-type-specifier
| elaborated-type-specifier
| typename-specifier
| cv-qualifier
;

type-specifier-seq:
type-specifier attribute-specifier-seq$_{opt}$
| type-specifier type-specifier-seq
;

trailing-type-specifier-seq:
trailing-type-specifier attribute-specifier-seq$_{opt}$
| trailing-type-specifier trailing-type-specifier-seq
;

simple-type-specifier:
nested-name-specifier$_{opt}$ type-name
| nested-name-specifier template simple-template-id
| char
| char16_t
| char32_t
| wchar_t
| bool
| short
| int
| long
| signed
| unsigned
| float
| double
| void
| auto
| decltype-specifier

;
type-name:
class-name
| enum-name
| typedef-name
| simple-template-id
;
decltype-specifier:
decltype (expression)
;
elaborated-type-specifier:
class-key attribute-specifier-seq$_{opt}$ nested-name-specifier$_{opt}$ identifier
| class-key nested-name-specifier$_{opt}$template$_{opt}$ simple-template-id
| enum nested-name-specifier$_{opt}$ identifier
;
enum-name:
identifier
;
enum-specifier:
enum-head enumerator-list$_{opt}$
| enum-head enumerator-list ,
;
enum-head:
enum-key attribute-specifier-seq$_{opt}$ identifier$_{opt}$ enum-base$_{opt}$
| enum-key attribute-specifier-seq$_{opt}$ nested-name-specifier identifier enum-base$_{opt}$
;
opaque-enum-declaration:
enum-key attribute-specifier-seq$_{opt}$ identifier enum-base$_{opt}$;
;
enum-key:
enum
| enum class
| enum struct
;
enum-base:
: type-specifier-seq
;
enumerator-list:
enumerator-definition
| enumerator-list , enumerator-definition
;
enumerator-definition:
enumerator
| enumerator = constant-expression
;

enumerator:
identifier
;
namespace-name:
original-namespace-name
| namespace-alias
;
original-namespace-name:
identifier
;
namespace-definition:
named-namespace-definition
| unnamed-namespace-definition
;
named-namespace-definition:
original-namespace-definition
| extension-namespace-definition
;
original-namespace-definition:
inline$_{opt}$ namespace identifier namespace-body
;
extension-namespace-definition:
inline$_{opt}$ namespace original-namespace-name namespace-body
;
unnamed-namespace-definition:
inline$_{opt}$ namespace namespace-body
;
namespace-body:
declaration-seq$_{opt}$
;
namespace-alias:
identifier
;
namespace-alias-definition:
namespace identifier = qualified-namespace-specifier ;
;
qualified-namespace-specifier:
nested-name-specifier$_{opt}$ namespace-name
;
using-declaration:
using typename$_{opt}$ nested-name-specifier unqualified-id ;
| using :: unqualified-id ;
;
using-directive:
using namespace ::$_{opt}$ nested-name-specifier$_{opt}$ namespace-name ;
;

asm-definition:
    asm (string-literal) ;
    ;
linkage-specification:
    extern string-literal declaration-seq$_{opt}$
    | extern string-literal declaration
    ;
init-declarator-list:
    init-declarator
    | init-declarator-list , init-declarator
    ;
init-declarator:
    declarator initializer$_{opt}$
    ;
declarator:
    ptr-declarator
    | noptr-declarator parameters-and-qualifiers trailing-return-type
    ;
ptr-declarator:
    noptr-declarator
    | ptr-operator ptr-declarator
    ;
noptr-declarator:
    declarator-id attribute-specifier-seq$_{opt}$
    | noptr-declarator parameters-and-qualifiers
    | noptr-declarator [constant-expression$_{opt}$] attribute-specifier-seq$_{opt}$
    | (ptr-declarator)
    ;
parameters-and-qualifiers:
    (parameter-declaration-clause) attribute-specifier-seq$_{opt}$ cv-qualifier-seq$_{opt}$
ref-qualifier$_{opt}$ exception-specification$_{opt}$
    ;
trailing-return-type:
    $-$ > trailing-type-specifier-seq abstract-declarator$_{opt}$
    ;
ptr-operator:
    * attribute-specifier-seq$_{opt}$ cv-qualifier-seq$_{opt}$
    | & attribute-specifier-seq$_{opt}$
    | && attribute-specifier-seq$_{opt}$
    | nested-name-specifier * attribute-specifier-seq$_{opt}$ cv-qualifier-seq$_{opt}$
    ;
cv-qualifier-seq:
    cv-qualifier cv-qualifier-seq$_{opt}$
    ;
cv-qualifier:
    const

| volatile
;
ref-qualifier:
&
| &&
;
declarator-id:
...$_{opt}$ id-expression
| nested-name-specifier$_{opt}$ class-name
;
type-id:
type-specifier-seq abstract-declarator$_{opt}$
;
abstract-declarator:
ptr-abstract-declarator
| noptr-abstract-declarator$_{opt}$ parameters-and-qualifiers trailing-return-type
| abstract-pack-declarator
;
ptr-abstract-declarator:
noptr-abstract-declarator
| ptr-operator ptr-abstract-declarator$_{opt}$
;
noptr-abstract-declarator:
noptr-abstract-declarator$_{opt}$ parameters-and-qualifiers
| noptr-abstract-declarator$_{opt}$ [constant-expression$_{opt}$] attribute-specifier-seq$_{opt}$
| (ptr-abstract-declarator)
;
abstract-pack-declarator:
noptr-abstract-pack-declarator
| ptr-operator abstract-pack-declarator
;
noptr-abstract-pack-declarator:
noptr-abstract-pack-declarator parameters-and-qualifiers
| noptr-abstract-pack-declarator [constant-expression$_{opt}$] attribute-specifier-seq$_{opt}$
| ...
;
parameter-declaration-clause:
parameter-declaration-list$_{opt}$ ...$_{opt}$
| parameter-declaration-list , ...
;
parameter-declaration-list:
parameter-declaration
| parameter-declaration-list , parameter-declaration
;

parameter-declaration:
attribute-specifier-seq$_{opt}$ decl-specifier-seq declarator
| attribute-specifier-seq$_{opt}$ decl-specifier-seq declarator = initializer-clause
| attribute-specifier-seq$_{opt}$ decl-specifier-seq abstract-declarator$_{opt}$
| attribute-specifier-seq$_{opt}$ decl-specifier-seq abstract-declarator$_{opt}$ = initializer-clause
;
function-definition:
attribute-specifier-seq$_{opt}$ decl-specifier-seq$_{opt}$ declarator virt-specifier-seq$_{opt}$ function-body
;
function-body:
ctor-initializer$_{opt}$ compound-statement
| function-try-block
| = default ;
|= delete ;
;
initializer:
brace-or-equal-initializer
| (expression-list)
;
brace-or-equal-initializer:
= initializer-clause
| braced-init-list
;
initializer-clause:
assignment-expression
| braced-init-list
;
initializer-list:
initializer-clause ...$_{opt}$
| initializer-list , initializer-clause ...$_{opt}$
;
braced-init-list:
initializer-list ,$_{opt}$
|
;

D.4 C++ Class

class-name:
identifier
| simple-template-id
;
class-specifier:
class-head member-specification$_{opt}$

;

class-head:

class-key attribute-specifier-seq$_{opt}$ class-head-name class-virt-specifier$_{opt}$ base-clause$_{opt}$

| class-key attribute-specifier-seq$_{opt}$ base-clause$_{opt}$

;

class-head-name:

nested-name-specifier$_{opt}$ class-name

;

class-virt-specifier:

final

;

class-key:

class

| struct

| union

;

member-specification:

member-declaration member-specification$_{opt}$

| access-specifier : member-specification$_{opt}$

;

member-declaration:

attribute-specifier-seq$_{opt}$ decl-specifier-seq$_{opt}$ member-declarator-list$_{opt}$;

| function-definition ;$_{opt}$

| using-declaration

| static_assert-declaration

| template-declaration

| alias-declaration

;

member-declarator-list:

member-declarator

| member-declarator-list , member-declarator

;

member-declarator:

declarator virt-specifier-seq$_{opt}$ pure-specifier$_{opt}$

| declarator brace-or-equal-initializer$_{opt}$

| identifier$_{opt}$ attribute-specifier-seq$_{opt}$: constant-expression

;

virt-specifier-seq:

virt-specifier

| virt-specifier-seq virt-specifier

;

virt-specifier:

override

| final

;

pure-specifier:
    = 0
    ;

D.5 C++ Derived classes

base-clause:
    : base-specifier-list
    ;
base-specifier-list:
    base-specifier ...$_{opt}$
    | base-specifier-list , base-specifier ...$_{opt}$
    ;
base-specifier:
    attribute-specifier-seq$_{opt}$ base-type-specifier
    | attribute-specifier-seq$_{opt}$ virtual access-specifier$_{opt}$ base-type-specifier
    | attribute-specifier-seq$_{opt}$ access-specifier virtual$_{opt}$ base-type-specifier
    ;
class-or-decltype:
    nested-name-specifier$_{opt}$ class-name
    | decltype-specifier
    ;
base-type-specifier:
    class-or-decltype
    ;
access-specifier:
    private
    | protected
    | public
    ;

D.6 C++ Special member functions

conversion-function-id:
    operator conversion-type-id
    ;
conversion-type-id:
    type-specifier-seq conversion-declarator$_{opt}$
    ;
conversion-declarator:
    ptr-operator conversion-declarator$_{opt}$
    ;
ctor-initializer:
    : mem-initializer-list
    ;
mem-initializer-list:

mem-initializer ...$_{opt}$
| mem-initializer , mem-initializer-list ...$_{opt}$
;
mem-initializer:
mem-initializer-id (expression-list$_{opt}$)
| mem-initializer-id braced-init-list
;
mem-initializer-id:
class-or-decltype
| identifier
;

D.7 C++ Overloading

operator-function-id:
OPERATOR operator
;
operator:
new
| delete
| +
| -
| !
| =
| ^ =
| &=
| <=
| >=
| ()
| []
| new[]
| *
| <
| |=
| &&
| delete[]
| /
| >
| <<
| ||
| %
| +=
|>>
| ++
| ^
| -=

| >>=
| −−
| &
| *=
| <<=
| ,
| $$
| /=
| ==
| − > *
| ∼
| %=
| !=
| − >
;

literal-operator-id:
OPERATOR "" identifier
;

D.8 C++ Templates

template-declaration:
template < template-parameter-list > declaration
;

template-parameter-list:
template-parameter
| template-parameter-list , template-parameter
;

template-parameter:
type-parameter
| parameter-declaration
;

type-parameter:
class ...$_{opt}$ identifier$_{opt}$
| class identifier$_{opt}$ = type-id
| typename ...$_{opt}$ identifier$_{opt}$
| typename identifier$_{opt}$ = type-id
| template < template-parameter-list > class ...$_{opt}$ identifier$_{opt}$
| template < template-parameter-list > class identifier$_{opt}$ = id-expression ;

simple-template-id:
template-name < template-argument-list$_{opt}$ >
;

template-id:
simple-template-id
| operator-function-id < template-argument-list$_{opt}$ >
| literal-operator-id < template-argument-list$_{opt}$ >

;
template-name:
identifier
;
template-argument-list:
template-argument ...$_{opt}$
| template-argument-list , template-argument ...$_{opt}$
;
template-argument:
constant-expression
| type-id
| id-expression
;
typename-specifier:
typename nested-name-specifier identifier
| typename nested-name-specifier template$_{opt}$ simple-template-id
;
explicit-instantiation:
extern$_{opt}$ template declaration
;
explicit-specialization:
template < > declaration
;

D.9 C++ Exception handling

try-block:
try compound-statement handler-seq
;
function-try-block:
try ctor-initializer$_{opt}$ compound-statement handler-seq
;
handler-seq:
handler handler-seq$_{opt}$
;
handler:
catch (exception-declaration) compound-statement
;
exception-declaration:
attribute-specifier-seq$_{opt}$ type-specifier-seq declarator
| attribute-specifier-seq$_{opt}$ type-specifier-seq abstract-declarator$_{opt}$
| ...
;
throw-expression:
throw assignment-expression$_{opt}$
;

exception-specification:
    dynamic-exception-specification
    | noexcept-specification
    ;
dynamic-exception-specification:
    throw (type-id-list$_{opt}$)
    ;
type-id-list:
    type-id ...$_{opt}$
    | type-id-list , type-id ...$_{opt}$
    ;
noexcept-specification:
    noexcept (constant-expression)
    | noexcept
    ;

D.10 C++ Basic concepts

translation-unit:
    declaration-seq$_{opt}$
    ;

E C grammar

E.1 C Expressions

primary-expression:
    identier
    | constant
    | string-literal
    | (expression)
    ;
postx-expression:
    primary-expression
    | postx-expression [expression]
    | postx-expression (argument-expression-list$_{opt}$)
    | postx-expression . identier
    | postx-expression − > identier
    | postx-expression ++
    | postx-expression −−
    | (type-name) initializer-list
    | (type-name) initializer-list ,
    ;
argument-expression-list:
    assignment-expression

```
    | argument-expression-list , assignment-expression
    ;
unary-expression:
    postx-expression
    | ++ unary-expression
    | −− unary-expression
    | unary-operator cast-expression
    | sizeof unary-expression
    | sizeof ( type-name )
    ;
unary-operator:
    &
    | *
    | +
    | -
    | ∼
    | !
    ;
cast-expression:
    unary-expression
    | ( type-name ) cast-expression
    ;
multiplicative-expression:
    cast-expression
    | multiplicative-expression * cast-expression
    | multiplicative-expression / cast-expression
    | multiplicative-expression % cast-expression
    ;
additive-expression:
    multiplicative-expression
    | additive-expression + multiplicative-expression
    | additive-expression - multiplicative-expression
    ;
shift-expression:
    additive-expression
    | shift-expression << additive-expression
    | shift-expression >> additive-expression
    ;
relational-expression:
    shift-expression
    | relational-expression < shift-expression
    | relational-expression > shift-expression
    | relational-expression <= shift-expression
    | relational-expression >= shift-expression
    ;
equality-expression:
```

```
    relational-expression
    | equality-expression == relational-expression
    | equality-expression != relational-expression
    ;
AND-expression:
    equality-expression
    | AND-expression & equality-expression
    ;
exclusive-OR-expression:
    AND-expression
    | exclusive-OR-expression ^ AND-expression
    ;
inclusive-OR-expression:
    exclusive-OR-expression
    | inclusive-OR-expression | exclusive-OR-expression
    ;
logical-AND-expression:
    inclusive-OR-expression
    | logical-AND-expression && inclusive-OR-expression
    ;
logical-OR-expression:
    logical-AND-expression
    | logical-OR-expression || logical-AND-expression
    ;
conditional-expression:
    logical-OR-expression
    | logical-OR-expression ? expression : conditional-expression
    ;
assignment-expression:
    conditional-expression
    | unary-expression assignment-operator assignment-expression
    ;
assignment-operator:
    =
    | *=
    | /=
    | %=
    | +=
    | -=
    | <<=
    | >>=
    | &=
    | ^ =
    | |=
    ;
expression:
```

assignment-expression
| expression , assignment-expression
;
constant-expression:
conditional-expression
;

E.2 C-Declarations

declaration:
declaration-speciers init-declarator-list$_{opt}$;
;
declaration-speciers:
storage-class-specier declaration-speciers$_{opt}$
| type-specier declaration-speciers$_{opt}$
| type-qualier declaration-speciers$_{opt}$
| function-specier declaration-speciers$_{opt}$
;
init-declarator-list:
init-declarator
| init-declarator-list , init-declarator
;
init-declarator:
declarator
| declarator = initializer
;
storage-class-specier:
typedef
| extern
| static
| auto
| register
;
type-specier:
void
| char
| short
| int
| long
| float
| double
| signed
| unsigned
| _Bool
| _Complex
| _Imaginary

| struct-or-union-specier
| enum-specier
| typedef-name
;

struct-or-union-specier:
struct-or-union identier$_{opt}$ { struct-declaration-list }
| struct-or-union identier
;

struct-or-union:
struct
| union
;

struct-declaration-list:
struct-declaration
| struct-declaration-list struct-declaration
;

struct-declaration:
specier-qualier-list struct-declarator-list ;
;

specier-qualier-list:
type-specier specier-qualier-list$_{opt}$
| type-qualier specier-qualier-list$_{opt}$
;

struct-declarator-list:
struct-declarator
| struct-declarator-list , struct-declarator
;

struct-declarator:
declarator
| declarator$_{opt}$: constant-expression
;

enum-specier:
enum identier$_{opt}$ { enumerator-list }
| enum identier$_{opt}$ { enumerator-list , }
| enum identier
;

enumerator-list:
enumerator
| enumerator-list , enumerator
;

enumerator:
enumeration-constant
| enumeration-constant = constant-expression
;

type-qualier:
const

| restrict
| volatile
;
function-specier:
inline
;
declarator:
pointer$_{opt}$ direct-declarator
;
direct-declarator:
identier
| (declarator)
| direct-declarator [type-qualier-list$_{opt}$ assignment-expression$_{opt}$]
| direct-declarator [static type-qualier-list$_{opt}$ assignment-expression]
| direct-declarator [type-qualier-list static assignment-expression]
| direct-declarator [type-qualier-list$_{opt}$ *]
| direct-declarator (parameter-type-list)
| direct-declarator (identier-list$_{opt}$)
;
pointer:
* type-qualier-list$_{opt}$
| * type-qualier-list$_{opt}$ pointer
;
type-qualier-list:
type-qualier
| type-qualier-list type-qualier
;
parameter-type-list:
parameter-list
| parameter-list , ...
;
parameter-list:
parameter-declaration
| parameter-list , parameter-declaration
;
parameter-declaration:
declaration-speciers declarator
| declaration-speciers abstract-declarator$_{opt}$
;
identier-list:
identier
identier-list , identier
;
type-name:
specier-qualier-list abstract-declarator$_{opt}$
;

abstract-declarator:
    pointer
    | pointer$_{opt}$ direct-abstract-declarator
    ;
direct-abstract-declarator:
    (abstract-declarator)
    | direct-abstract-declarator$_{opt}$ [assignment-expression$_{opt}$]
    | direct-abstract-declarator$_{opt}$ [*]
    | direct-abstract-declarator$_{opt}$ (parameter-type-list$_{opt}$)
    ;
typedef-name:
    identier
    ;
initializer:
    assignment-expression
    | initializer-list
    | initializer-list ,
    ;
initializer-list:
    designation$_{opt}$ initializer
    | initializer-list , designation$_{opt}$ initializer
    ;
designation:
    designator-list =
    ;
designator-list:
    designator
    | designator-list designator
    ;
designator:
    [constant-expression]
    | . identier
    ;

E.3 C-Statements

statement:
    labeled-statement
    | compound-statement
    | expression-statement
    | selection-statement
    | iteration-statement
    | jump-statement
    ;
labeled-statement:
    identier : statement

| case constant-expression : statement
| default : statement
;
compound-statement:
block-item-list$_{opt}$
;
block-item-list:
block-item
| block-item-list block-item
;
block-item:
declaration
| statement
;
expression-statement:
expression$_{opt}$;
selection-statement:
if (expression) statement
| if (expression) statement else statement
| switch (expression) statement
;
iteration-statement:
while (expression) statement
| do statement while (expression) ;
| for (expression$_{opt}$; expression$_{opt}$; expression$_{opt}$) statement
| for (declaration expression$_{opt}$; expression$_{opt}$) statement
;
jump-statement:
goto identier ;
| continue ;
| break ;
| return expression$_{opt}$;
;

E.4 C-External denitions

translation-unit:
external-declaration
| translation-unit external-declaration
;
external-declaration:
function-denition
| declaration
;
function-denition:
declaration-speciers declarator declaration-list$_{opt}$ compound-statement

;
declaration-list:
declaration
| declaration-list declaration
;

F Preprocessing

preprocessing-file:
group$_{opt}$
;
group:
group-part
| group group-part
;
group-part:
if-section
| control-line
| text-line
| # non-directive
;
if-section:
if-group elif-groups$_{opt}$ else-group$_{opt}$ endif-line
;
if-group:
# if constant-expression new-line group$_{opt}$
| # ifdef identifier new-line group$_{opt}$
| # ifndef identifier new-line group$_{opt}$
;
elif-groups:
elif-group
| elif-groups elif-group
;
elif-group:
# elif constant-expression new-line group$_{opt}$
;
else-group:
# else new-line group$_{opt}$
;
endif-line:
endif new-line
;
control-line:
include header-name new-line
define identifier replacement-list new-line

define identifier lparen identifier-list$_{opt}$) replacement-list new-line
define identifier lparen ...) replacement-list new-line
define identifier lparen identifier-list, ...) replacement-list new-line
undef identifier new-line
line anything new-line
error anything new-line
pragma anything new-line
new-line
;

text-line:
pp-tokens$_{opt}$ new-line
;

non-directive:
pp-tokens new-line
;

lparen:
a (character not immediately preceded by white-space
;

identifier-list:
identifier
| identifier-list , identifier
;

replacement-list:
pp-tokens$_{opt}$
;

header-name:
¡ h-char-sequence ¿
| " q-char-sequence "
;

h-char-sequence:
h-char
| h-char-sequence h-char
;

h-char:
any member of the source character set except the new-line character and ¿
;

q-char-sequence:
q-char
| q-char-sequence q-char
;

q-char:
any member of the source character set except the new-line character and "
;

References

[1] Ritchie and Kernighan. *The C Programming Language*. Prentice Hall, 1988.

[2] Bjarne Stroustrup. *The Design and Evolution of C++*. Massachusetts, 1994.

[3] E. Leontiadis V.V. Dimakopoulos and G. Tzoumas. A portable c compiler for OpenMP v.2.0. In *Proc. of the 5th European Workshop on OpenMP (EWOMP)*, pages 5–11, 2003.

[4] Michael D. Ernst, Greg J. Badros, and David Notkin. An empirical analysis of c preprocessor use. *IEEE Trans. Softw. Eng.*, 28(12):1146–1170, December 2002.

[5] Yoann Padioleau. Parsing c/c++ code without pre-processing. In *Proceedings of the 18th International Conference on Compiler Construction: Held As Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*, CC '09, pages 109–125, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] `http://www.open-std.org/jtc1/sc22/wg21/`.