GCC Internals – Generic, Gimple, and RTL

J. Trávníček

Faculty of Information Technology
Czech Technical University in Prague

GEN – 4

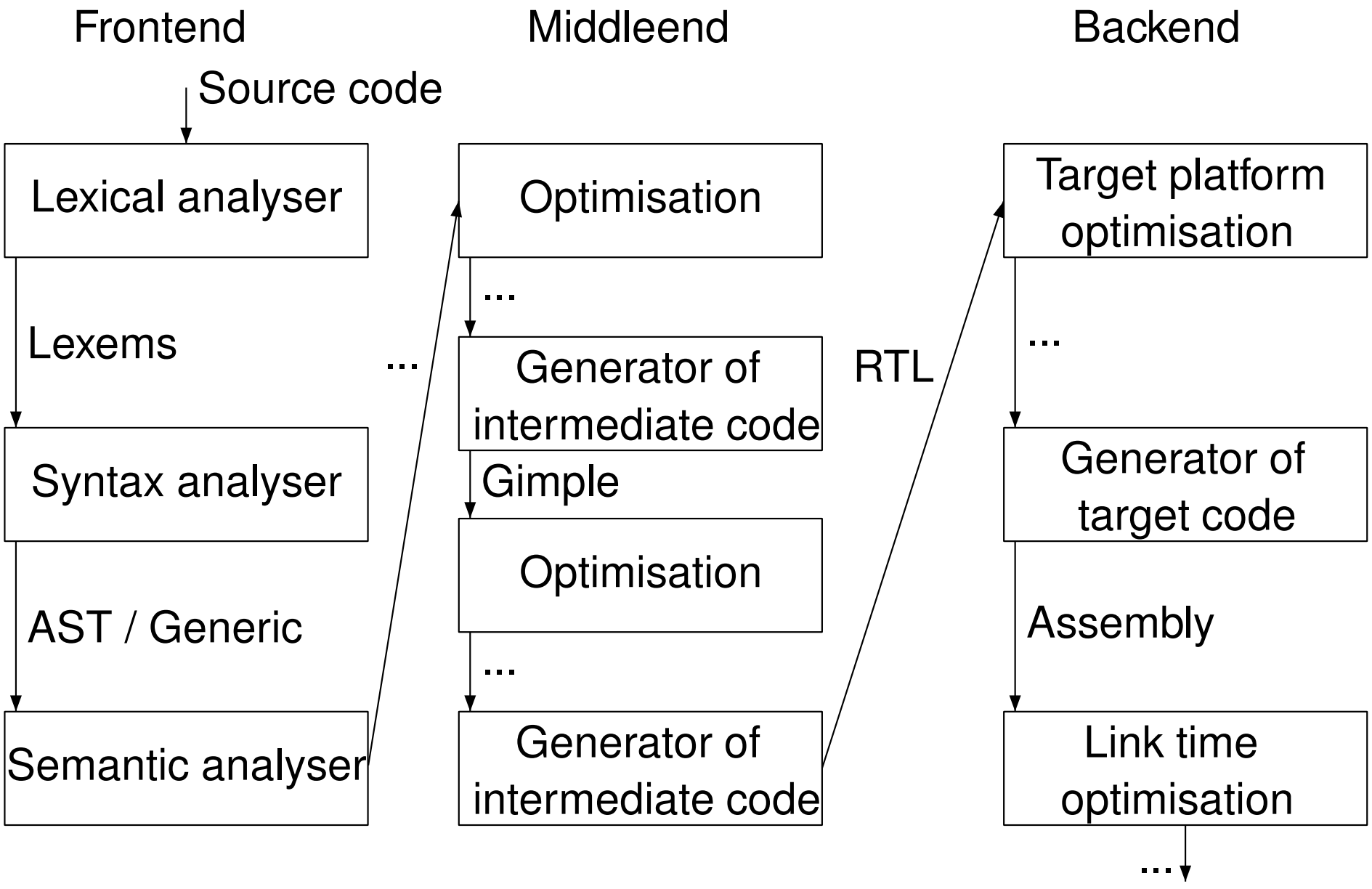
Outline

1. Introduction
   - Compiler Design
2. Generic
   - Generic
3. Gimple
   - Gimple
4. RTL
   - RTL
5. Live example
   - Live example

Outline

1. Introduction
   - Compiler Design
2. Generic
   - Generic
3. Gimple
   - Gimple
4. RTL
   - RTL
5. Live example
   - Live example

Compiler outline



Intermediate representations used in GCC

- Generic,
- Gimple,
- Register Transfer Language (RTL).

Outline

1 Introduction
- Compiler Design

2 Generic
- Generic

3 Gimple
- Gimple

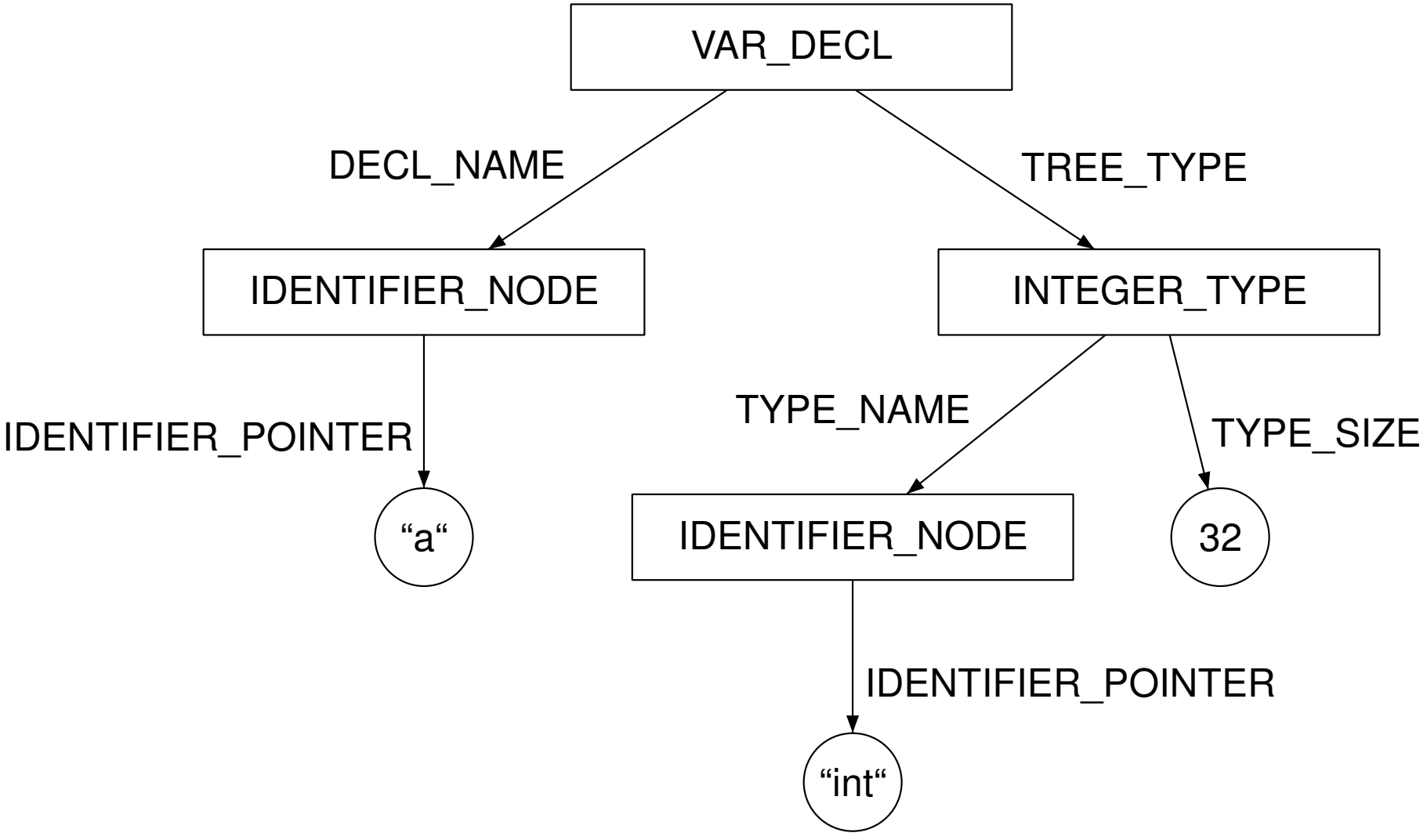
4 RTL
- RTL

5 Live example
- Live example

Introduction

- Tree representation.
- Internally union.
- Constructed by functions make_node, build_decl, build_int_cst, ..., build1, build2, ...
- Language independent.
- Represent entire functions, global definitions, ...
- Mostly close to C/C++ programming language.

Generic representation of variable declaration



Nodes of generic – binary

- EQ_EXPR, NE_EXPR
- LT_EXPR, LE_EXPR
- GT_EXPR, GE_EXPR
- TRUTH_AND_EXPR
- TRUTH_ANDIF_EXPR
- TRUTH_OR_EXPR
- TRUTH_ORIF_EXPR
- TRUTH_XOR_EXPR
- PLUS_EXPR, MINUS_EXPR
- MULT_EXPR, TRUNC_DIV_EXPR

```
build2(EQ_EXPR, integer_type_node, op1, op2)
```

Nodes of generic – unary

- ABS_EXPR
- NEGATE_EXPR
- TRUTH_NOT_EXPR

```
build1(NEGATE_EXPR, integer_type_node, op1);
```

Nodes of generic – {pre, post}{inc, dec}

- PREDECREMENT_EXPR
- POSTDECREMENT_EXPR
- PREINCREMENT_EXPR
- POSTINCREMENT_EXPR

```
build2(PREDECREMENT_EXPR, integer_type_node, op1, by);
```

The second argument is how much to increment or decrement by. For a pointer, it would be the size of the object pointed to.

Nodes of generic – pointers

reference

```
build1(ADDR_EXPR, build_pointer_type(TREE_TYPE(op1)), op1);
```

dereference

```
build1(INDIRECT_REF, TREE_TYPE(TREE_TYPE(op1)), op1);
```

array indexing

```
build4(ARRAY_REF, TREE_TYPE(TREE_TYPE(array)), array, index,
  NULL_TREE, NULL_TREE);
```

Nodes of generic – variables

```
tree declaration = build_decl (UNKNOWN_LOCATION, VAR_DECL,
  get_identifier(name), type);
TREE_ADDRESSABLE(declaration) = true;
TREE_USED(declaration) = true;
DECL_INITIAL(declaration) = ... (expr);

tree declaration = build_decl (UNKNOWN_LOCATION, VAR_DECL,
  get_identifier(name), type);
TREE_ADDRESSABLE(declaration) = true;
TREE_USED(declaration) = true;
TREE_STATIC(declaration) = true;
TREE_PUBLIC(declaration) = true;
DECL_INITIAL(declaration) = ... (expr);
```

Nodes of generic – function type

```
tree params = NULL_TREE;
params = chainon( params,
  tree_cons (NULL_TREE, integer_type_node, NULL_TREE));
tree resdecl = build_decl (BUILTINS_LOCATION, RESULT_DECL,
  NULL_TREE, integer_type_node);
tree fntype = build_function_type(TREE_TYPE(resdecl), params);
```

Nodes of generic – function declaration

```
tree param_decl = NULL_TREE;
tree number = build_decl (UNKNOWN_LOCATION, PARM_DECL,
  get_identifier("number"), integer_type_node);
DECL_ARG_TYPE(number) = integer_type_node;
param_decl = chainon( param_decl, number );

tree fndecl = build_decl( UNKNOWN_LOCATION, FUNCTION_DECL,
  get_identifier("factorial"), ... (fntype) );
DECL_ARGUMENTS(fndecl) = param_decl;
DECL_RESULT( fndecl ) = resdecl;
TREE_STATIC(fndecl) = true;
TREE_PUBLIC( fndecl ) = true;

DECL_INITIAL(fndecl) = ... (block);
DECL_SAVED_TREE(fndecl) = ... (bind);
```

Nodes of generic – blocks

```
tree decls = NULL_TREE;
tree variable_i = build_decl (...);
decls = chainon(decls, variable_i);

tree block = build_block(decls, NULL_TREE,
  NULL_TREE, NULL_TREE);
TREE_USED(block) = true;

tree stmts = alloc_stmt_list ();
append_to_statement_list(... (expr), &stmts);

tree bind = build3( BIND_EXPR, void_type_node,
  BLOCK_VARS(block), stmts, block);
TREE_SIDE_EFFECTS(bind) = true;

BLOCK_SUPERCONTEXT(block) = fndecl;
```

Nodes of generic – nested block with variables

```
tree decls = NULL_TREE;
tree variable_k = build_decl (...);
decls = chainon(decls, variable_k);

tree stmts = alloc_stmt_list ();
append_to_statement_list(... (expr), &stmts2);

tree bind = build3( BIND_EXPR, void_type_node,
  decls, stmts, NULL_TREE );
TREE_SIDE_EFFECTS(bind) = true;
```

Nodes of generic – nested block without variables

- Intuitively no need for bind expression – no variables to bind to statement list,
- moreover, not even block is needed.
- Conclusion, sequence of statements can be represented with statement list only.

Nodes of generic – modify and return expression

modify

```
build2(MODIFY_EXPR, TREE_TYPE(op1Generic),
  op1Generic, op2Generic);
```

return

```
build1(RETURN_EXPR, void_type_node, op1Generic);
```

Nodes of generic – call expression

```
tree * args_vec = XNEWVEC( tree, argumentsSize );
int i = 0;
for(argument : arguments) {
  args_vec[i++] = ... (expr);
}

tree fndecl = ...;

tree call = build_call_expr_loc_array( UNKNOWN_LOCATION,
  fndecl, argumentsSize, args_vec );
SET_EXPR_LOCATION(call, UNKNOWN_LOCATION);
TREE_USED(call) = true;
```

Nodes of generic – loops, if

loop

```
build1(LOOP_EXPR, void_type_node, ... (bind));
```

exit

```
build1(EXIT_EXPR, void_type_node, ... (expr));
```

if

```
build3(COND_EXPR, void_type_node,
  ... (expr), ... (bind), ... (bind));
```

- Nodes are defined in tree.def file,
- frontend can introduce new nodes,
- generic must be translated to gimple for middle-end,
- frontend must provide function to translate new nodes to gimple,
- generic can be dumped during compilation to file with fdump-tree-original-raw,
- c-like representation of generic can be dumped with fdump-tree-original.

Outline

1 Introduction
- Compiler Design

2 Generic
- Generic

3 Gimple
- Gimple

4 RTL
- RTL

5 Live example
- Live example

Introduction

- Derived from Generic,
- three-address representation,
- temporaries hold intermediate values,
- labels and conditioned jumps represent all control structures

Temporaries

- Generic complex expressions are split to more expressions:

$$a = b + c + d$$

become

$$T1 = b + c$$

$$a = T1 + d$$

Control statements – if

- Control statements are converted to labels and (conditioned) gotos:

```
if ( ... ) goto <D.103>; else goto <D.104>;
<D.103>:
{
}
goto <D.105>;
<D.104>:
{
}
<D.105>:
```

Control statements – loops

- Control statements are converted to labels and (conditioned) gotos:

```
<D.99>:
{
  ...
  if ( ... ) goto <D.107>; else goto <D.109>;
  <D.109>:
}
goto <D.99>;
<D.107>:
```

Internal form dump

- gimple can be dumped during compilation to file with fdump-tree-gimple.
- all stages of gimple processing can be dumped during compilation to file with fdump-tree-all.

Outline

1. Introduction
   - Compiler Design
2. Generic
   - Generic
3. Gimple
   - Gimple
4. RTL
   - RTL
5. Live example
   - Live example

Introduction

- Low level intermediate representation,
- algebraic representation of instructions,
- representation inspired by lisp.

Examples

- Constant: (const_int i),
- Register: (reg:m n); n – number of machine or pseudo register, m – mode (full word, half word, single byte, ...),
- Memory: (mem:m addr alias); m – memory size, addr – address, alias – reference alias,
- Addition: (plus:m x y); x, y – arguments, m – mode,
- Preincrement: (pre_inc:m x), ...,
- Equal: (eq:m x y), ...,
- Call: (call (mem:fm addr) nbytes); nbytes – number of bytes of arguments, fm – mode, addr – address of subroutine,
- ...

Examples

- all stages of rtl processing can be dumped with fdump-rtl-all.

Outline

1. Introduction
   - Compiler Design
2. Generic
   - Generic
3. Gimple
   - Gimple
4. RTL
   - RTL
5. Live example
   - Live example

Live example

...