Solidity_language is used to read in solidity and convert it to C++ mode, solidity_grammar defines an enumeration of types for type determination and conversion, solidity_convert is responsible for specific conversions and handles traversals and conversions, solidity_convert_ literals is specifically responsible for converting integer boolean strings and hexadecimal, and pattern detects solidity code.

The solidity_language class uses solidity_convert to convert ASTs. solidity_convert relies on solidity_grammar to understand the structure of the Solidity code it is converting. solidity_convert_literal supports converting literals and is used by solidity_convert.

Solidity_convert

nlohmann::json &ast_json; // json for Solidity AST. Use vector for multiple contracts

bool solidity_convertert::convert(){
Perform pattern-based verification
Populate the context with symbols annotated based on each AST node, and hence prepare for the GOTO conversion.**}**
Using:
convert_ast_nodes：Iterate over and then process the nodes in the JSON AST tree.
get_noncontract_defition：Non-contractual definitions.
get_struct_class：Structure Definition.
add_implicit_constructor：Responsible for adding an implicit constructor if one is not explicitly defined in the contract.
multi_transaction_verification: Multi-transaction validation for –contract.
multi_contract_verification: Validate multiple contracts across the file.

bool convert_ast_nodes(const nlohmann::json &contract_def);
Using:
get_decl: used to process each declaration node in the AST. get decl in rule contract-body-element. get decl in rule variable-declaration-statement, e.g. function local declaration.

bool get_decl(const nlohmann::json &ast_node, exprt &new_expr);
using:
bool get_var_decl(const nlohmann::json &ast_node, exprt &new_expr); Deals with variable declarations
bool get_function_definition(const nlohmann::json &ast_node) //Deals with function definitions
bool get_struct_class(const nlohmann::json &ast_node); Deals with structure declarations
bool get_error_definition(const nlohmann::json &ast_node); Deals with error statements

//Handling implicit constructors

1. add_implicit_constructor()//Adding an implicit constructor to a class or structure
2. get_implicit_ctor_call(exprt &new_expr, const std::string &contract_name)// Getting the invocation expression of an implicit constructor
3. get_struct_class_fields(const nlohmann::json &ast_node, struct_typet &type)// Parses the fields in the structure definition and adds those fields to the internal type representation
4. get_struct_class_method(const nlohmann::json &ast_node, struct_typet &type)// Parses the methods defined in the structure and processes them accordingly.
5. get_access_from_decl(const nlohmann::json &ast_node, struct_typet::componentt &comp)// Extract the access rights information from the declaration and set the appropriate attributes
6. get_block(const nlohmann::json &expr, exprt &new_expr)// Parsing Code Blocks Consisting of Multiple Statements.
7. get_statement(const nlohmann::json &block, exprt &new_expr)
8. get_binary_operator_expr(const nlohmann::json &expr, exprt &new_expr)//Parsing binary arithmetic expressions.
9. get_compound_assign_expr(const nlohmann::json &expr, exprt &new_expr)// Parsing Compound Assignment Expressions
10. get_unary_operator_expr(const nlohmann::json &expr, const nlohmann::json &literal_type, exprt &new_expr)// Parsing unary arithmetic expressions
11. get_conditional_operator_expr(const nlohmann::json &expr, exprt &new_expr)// Parsing Conditional Expressions
12. get_cast_expr(const nlohmann::json &cast_expr, exprt &new_expr, const nlohmann::json literal_type = nullptr)// Parsing Type Conversion Expressions
13. get_var_decl_ref(const nlohmann::json &decl, exprt &new_expr)// Getting references to variable declarations
14. get_func_decl_ref(const nlohmann::json &decl, exprt &new_expr)// Getting a reference to a function declaration
15. get_enum_member_ref(const nlohmann::json &decl, exprt &new_expr)// Getting a reference to an enumeration member
16. get_decl_ref_builtin(const nlohmann::json &decl, exprt &new_expr)// Getting references to built-in functions or variables
17. get_type_description(const nlohmann::json &type_name, typet &new_type)// Parsing type descriptions and converting to internal type representations
18. get_func_decl_ref_type(const nlohmann::json &decl, typet &new_type)// Getting information about the type of a function declaration
19. get_array_to_pointer_type(const nlohmann::json &decl, typet &new_type)// Converting an array type to a pointer type
20. get_elementary_type_name(const nlohmann::json &type_name, typet &new_type)// Get elementary type name
21. get_parameter_list(const nlohmann::json &type_name, typet &new_type)// Parsing the argument list of a function or method
22. get_state_var_decl_name(const nlohmann::json &ast_node, std::string &name, std::string &id)// Getting State Variable Declarations

23. get_var_decl_name(const nlohmann::json &ast_node, std::string &name, std::string &id)// Get the name of the variable declaration
24. get_function_definition_name(const nlohmann::json &ast_node, std::string &name, std::string &id)// Get the name of the function definition
25. get_constructor_call(const nlohmann::json &ast_node, exprt &new_expr)// Parsing Constructor Calls
26. get_current_contract_name(const nlohmann::json &ast_node, std::string &contract_name)// Get the name of the current contract
27. get_empty_array_ref(const nlohmann::json &ast_node, exprt &new_expr)// Handling empty array references

key function:

bool get_expr(const nlohmann::json &expr, exprt &new_expr);

Function overloading, as a proxy or interface simplifier.

bool get_expr(const nlohmann::json &expr, const nlohmann::json &expr_common_type, exprt &new_expr);

Populate the out parameter with the expression based on the solidity expression grammar. More specifically, parse each expression in the AST json and convert it to a exprt ("new_expr"). The expression may have sub-expression.

@param expr The expression that is to be converted to the IR

@param literal_type Type information ast to create the the literal

type in the IR (only needed for when the expression is a literal).

A literal_type is a "typeDescriptions" ast_node.

we need this due to some info is missing in the child node.

@param new_expr Out parameter to hold the conversion

@return true iff the conversion has failed

@return false iff the conversion was successful

Details:

locationt location;

get_start_location_from_stmt(expr, location);// Initialising location information

Get expression type.

```
SolidityGrammar::ExpressionT type = SolidityGrammar::get_expression_t(expr);
log_debug(
  "solidity",
  " @@@ got Expr: SolidityGrammar::ExpressionT::{}",
  SolidityGrammar::expression_to_str(type));
```

Type Branch Handling:

Binary Operator: BinaryOperatorClass

Unary operators: UnaryOperatorClass

Conditional operator: ConditionalOperatorClass

Declared Reference Expressions: DeclRefExprClass
Literal: literral
Tuple expression: Tuple
Function Call: CallExprClass
Implicit conversion expression: ImplicitCastExprClass
Index Access: IndexAccess
Create new object: NewExpression
Member Calls (including Contract, Struct, Enum Member Calls): ContractMemberCall, StructMemberCall, EnumMemberCall
Type name expression: ElementaryTypeNameExpression

// line number and locations. Processing location information in AST nodes

1. get_location_from_decl // Extracting location information from declared nodes
2. get_start_location_from_stmt // Extracting start position information from statement nodes
3. get_final_location_from_stmt // Extracting end position information from statement nodes
4. get_line_number // Extract line numbers based on AST nodes
5. add_offset// Add Offset。
6. get_src_from_json // some nodes may have "src" inside a member json object. we need to deal with them case by case based on the node type
7. move_symbol_to_context
8. multi_transaction_verification// Verify Multi-Trading Conditions
9. multi_contract_verification// Validating Multi-Contract Structures

//Helper function to handle conversions in Json files

1. std::string get_modulename_from_path(std::string path); Extract module name from file path。
2. std::string get_filename_from_path(std::string path); Extracting filenames from full paths。
3. const nlohmann::json &solidity_convertert::find_decl_ref(int ref_decl_id, std::string &contract_name)// find declaration reference
4. const nlohmann::json &find_constructor_ref(int ref_decl_id); Look up nodes in the AST。
5. void convert_expression_to_code(exprt &expr); Converting expression nodes into executable code form
6. bool check_intrinsic_function(const nlohmann::json &ast_node)// Check if it is a built-in function

// Type and expression handling

1. make_implicit_cast_expr//Create implicit type conversion expressions.
2. make_return_type_from_typet//Create return types from types.
3. make_pointee_type// Since Solidity function call node does not have enough information, we need to make a JSON object manually create a JSON object to complete the conversions of function to pointer decay
4. make_array_elementary_type// Function used to extract the type of the array and its elements
5. make_array_to_pointer_type // Function to replace the content of ["typeIdentifier"] with "ArrayToPtr"
6. get_array_size //Get the array size from the type description.

7. is_dyn_array//Determines if the array is dynamic.
8. add_dyn_array_size_expr//Add a size expression to a dynamic array type.

//literal conversion functions.
bool convert_integer_literal(
    const nlohmann::json &integer_literal,
    std::string the_value,
    exprt &dest); //integer conversion
bool convert_bool_literal(
    const nlohmann::json &bool_literal,
    std::string the_value,
    exprt &dest); //boolean conversion
bool convert_string_literal(std::string the_value, exprt &dest); //string conversion。
bool convert_hex_literal(std::string the_value, exprt &dest, const int n = 0); //hexadecimal conversion

//Auxiliary data structures:
Mapping from the Contract_id to the Contract_Name
  std::unordered_map<int, std::string> exportedSymbolsList;
Inheritance Order Record <contract_name, Contract_id>
  std::unordered_map<std::string, std::vector<int>> linearizedBaseList;
Store the ast_node["id"] of contract/struct/function/...
  std::unordered_map<int, std::string> scope_map;

Private：
Receive basic types from solidity and convert them
get_elementary_type_name_uint: unsigned integer
get_elementary_type_name_int: signed integer
get_elementary_type_name_bytesn: byte sequence

Solidity_grammar

```
enum ContractBodyElementT
{
   VarDecl = 0, // rule variable-declaration
   FunctionDef, // rule function-definition
   StructDef,      // rule struct-definition
   EnumDef,          // rule enum-definition
   ErrorDef,        // rule error-definition
   ContractBodyElementTError
};// In a similar way, enumeration types are created to express the different elements of a solidity contract.

//The following function gets the enumeration type first：
ContractBodyElementT get_contract_body_element_t(const nlohmann::json &element)
TypeNameT get_type_name_t(const nlohmann::json &type_name)
ElementaryTypeNameT get_elementary_type_name_t(const nlohmann::json &type_name)
ParameterListT get_parameter_list_t(const nlohmann::json &type_name)
BlockT get_block_t(const nlohmann::json &block)
StatementT get_statement_t(const nlohmann::json &stmt)
ExpressionT get_expression_t(const nlohmann::json &expr)
VarDeclStmtT get_var_decl_stmt_t(const nlohmann::json &stmt)
FunctionDeclRefT get_func_decl_ref_t(const nlohmann::json &decl)
ImplicitCastTypeT get_implicit_cast_type_t(std::string cast)

//Then the following functions convert the different enumeration types to strings：
const char* contract_body_element_to_str(ContractBodyElementT type)
const char* type_name_to_str(TypeNameT type)
const char* elementary_type_name_to_str(ElementaryTypeNameT type)
const char* parameter_list_to_str(ParameterListT type)
const char* block_to_str(BlockT type)
const char* statement_to_str(StatementT type)
const char* expression_to_str(ExpressionT type)
const char* var_decl_statement_to_str(VarDeclStmtT type)
const char* func_decl_ref_to_str(FunctionDeclRefT type)
const char* implicit_cast_type_to_str(ImplicitCastTypeT type)

//These functions calculate the type size：
unsigned int uint_type_name_to_size(ElementaryTypeNameT)
unsigned int int_type_name_to_size(ElementaryTypeNameT)
unsigned int bytesn_type_name_to_size(ElementaryTypeNameT)

//Operator acquisition, binary and unary
```

get_expression_t(const nlohmann::json &expr) // received expression
ExpressionT get_expr_operator_t(const nlohmann::json &expr) //binary
ExpressionT get_unary_expr_operator_t(const nlohmann::json &expr, bool uo_pre)//unary

VisibilityT get_access_t(const nlohmann::json &ast_node); // should be an access modifier

Solidity_convert_literals

convert_integer_literal
convert_bool_literal
convert_string_literal
convert_hex_literal
//Converts integers, booleans, strings and hexadecimal to the types used internally.

solidity_language.cpp

parse //Read the AST file
typecheck //check the syntax.
convert_intrinsics//Mapping solidity syntax to C++ for post-processing

pattern_check.cpp

safety check