GOTC 2023

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE，INTO THE FUTURE

「OpenSDV」专场

基于Clang Static Analyzer的车规安全检测工具 - zchecker

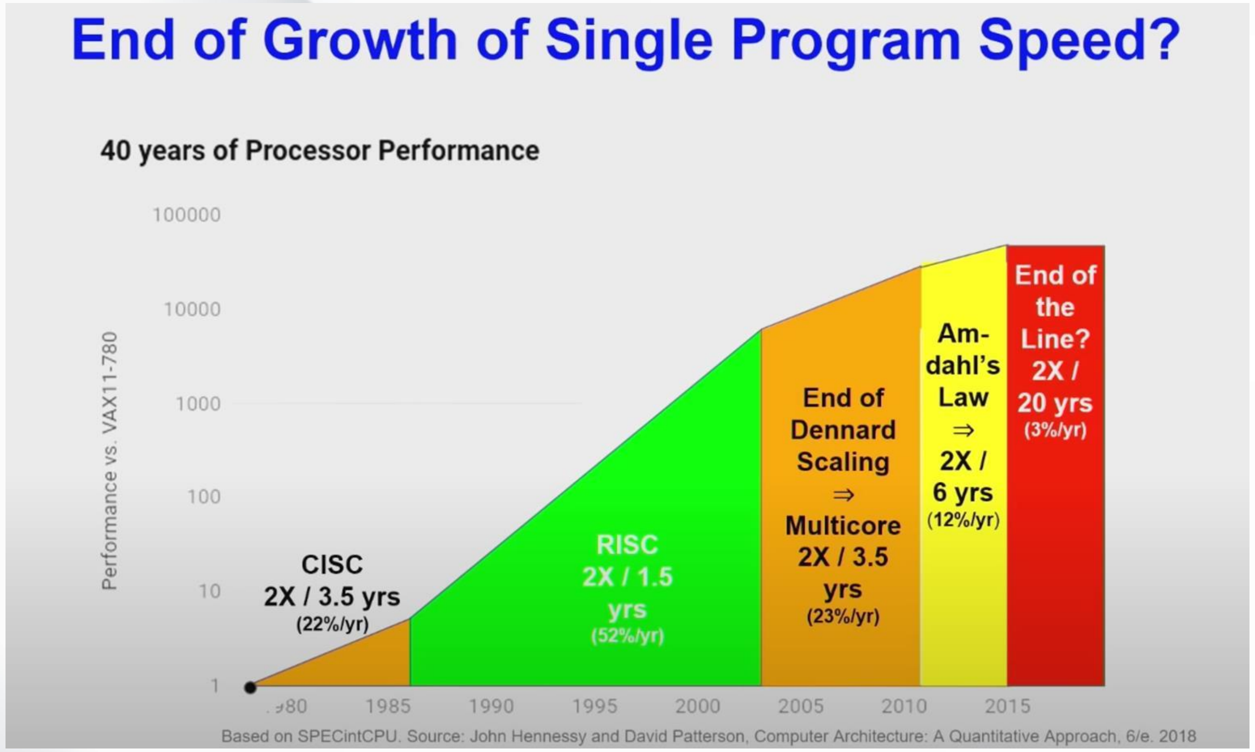
伍华林 2023年05月28日

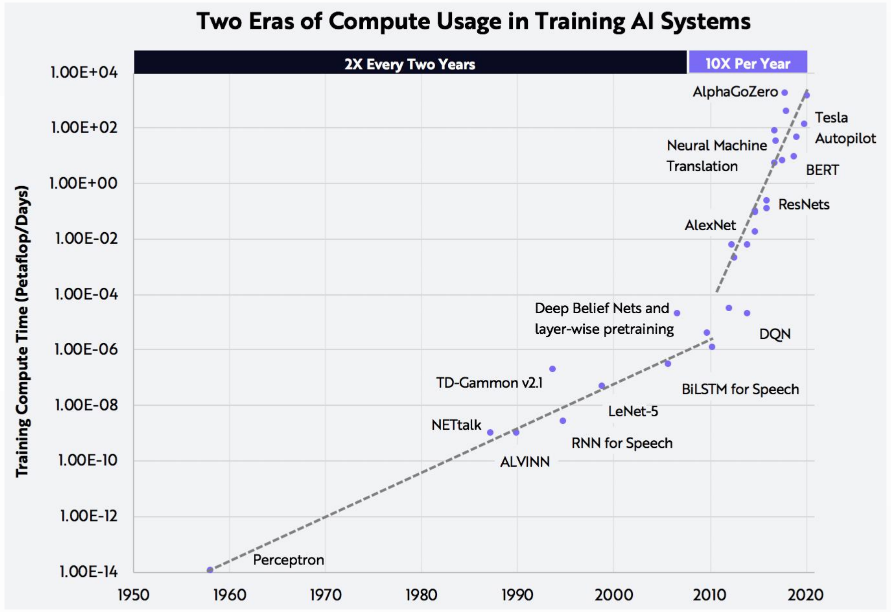
About me

A LLVM guy since 2008

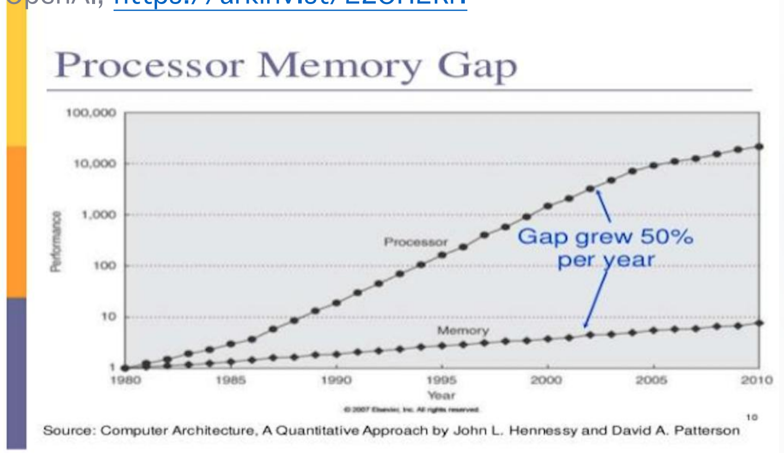
Moore's Law & Memory Wall

GOTC





Source: ARK Investment Management LLC, "AI and Compute." OpenAI, https://arkinv.st/2ZOH2Rr.



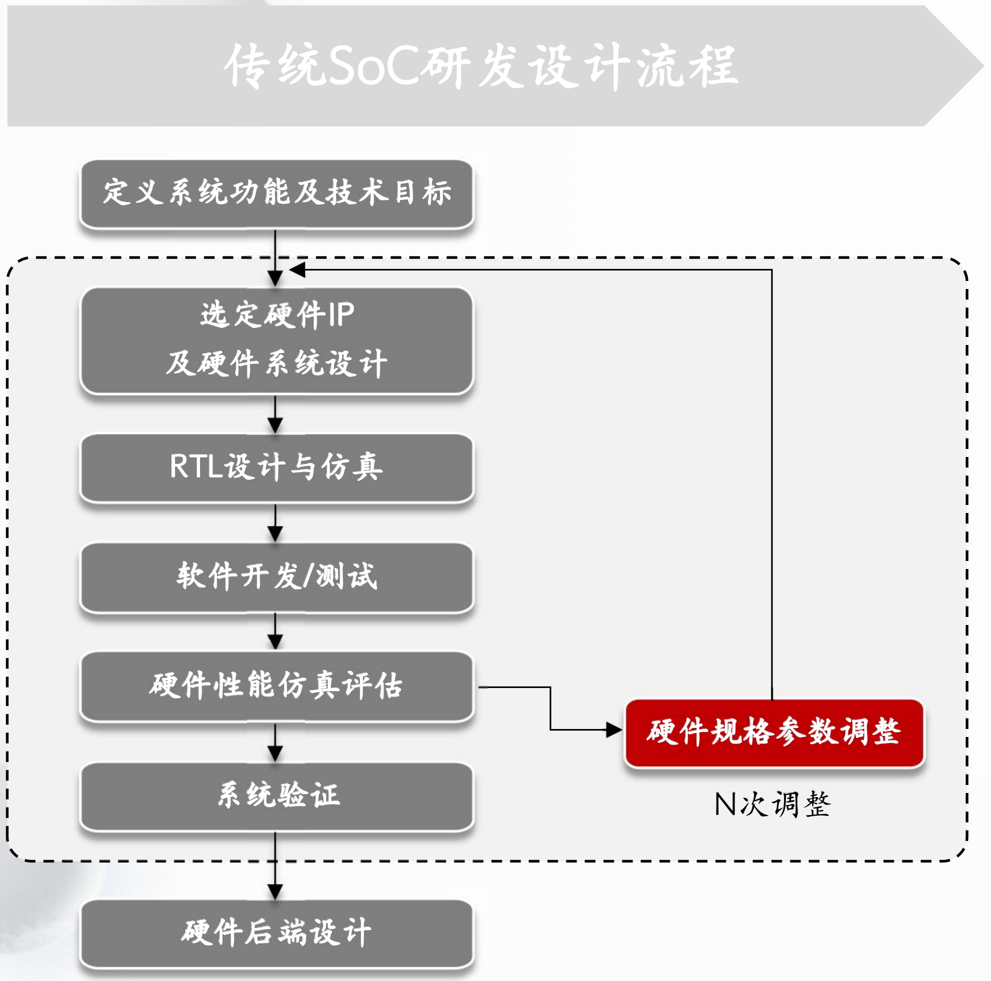
全球开源技术峰会
THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

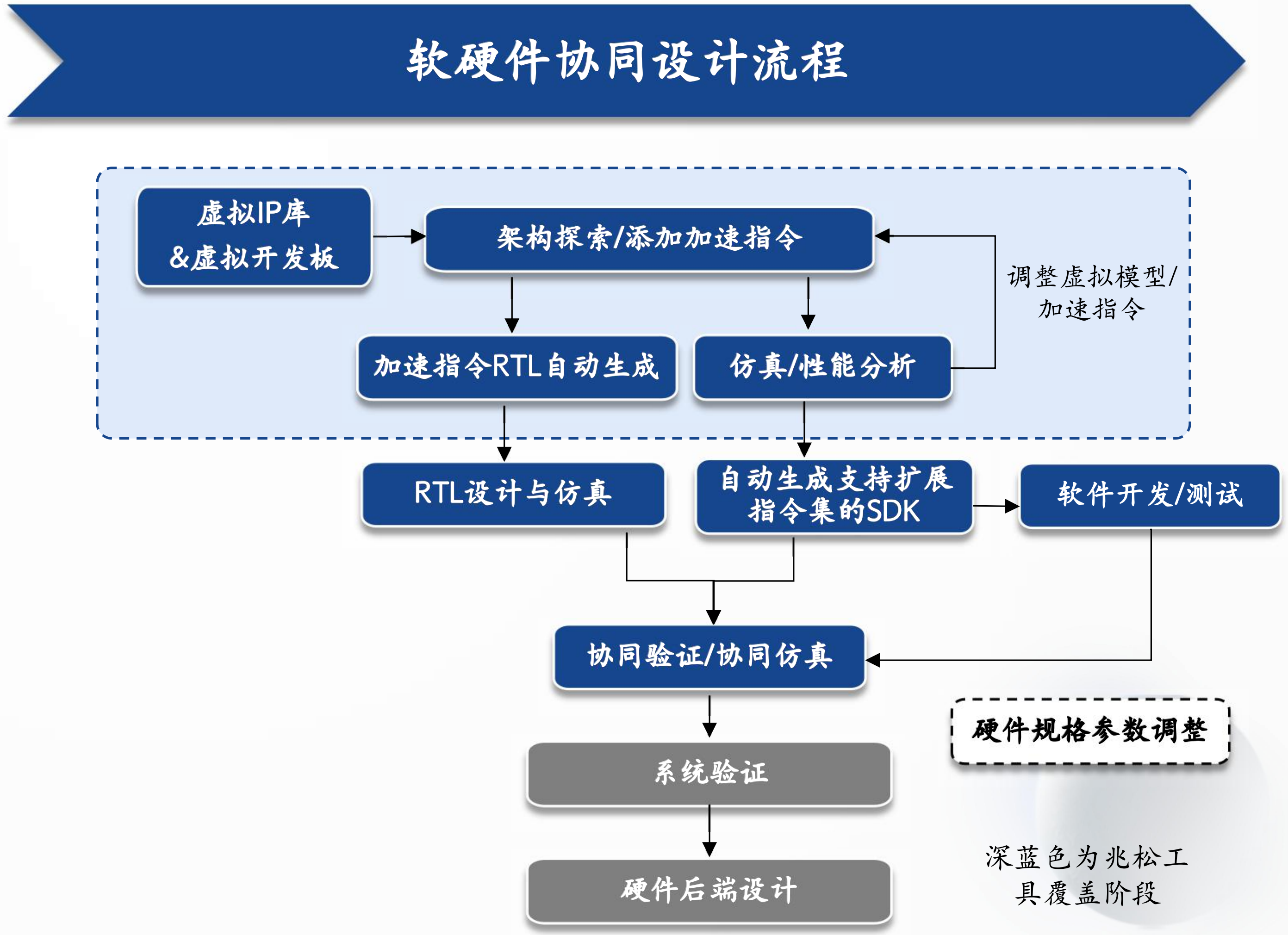
OPENSDV

TeraPines 兆松科技

兆松科技 – 软硬件协同设计服务提供商

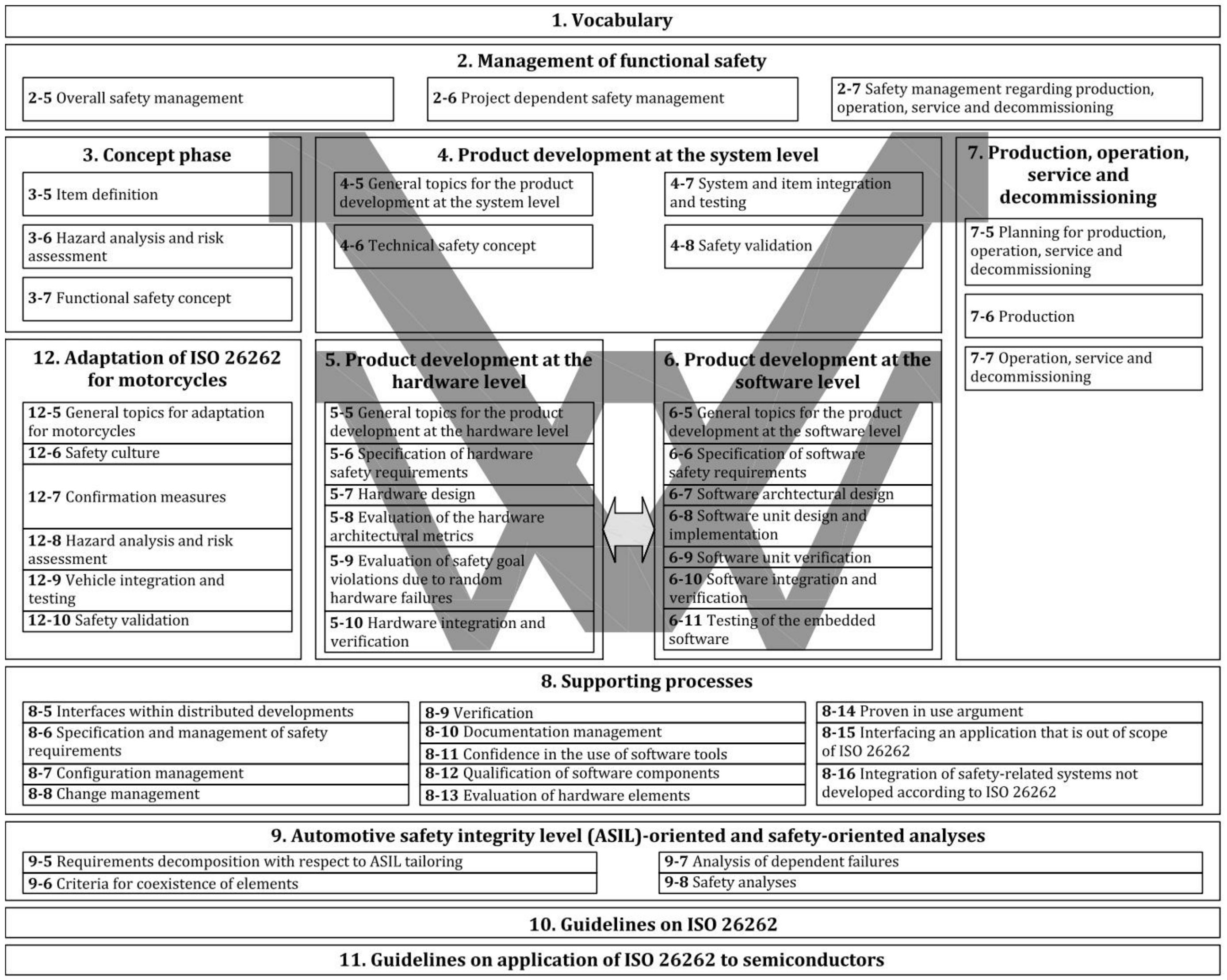
GOTC





更多内容B站搜索： RISC-V软硬件协同设计全流程软件栈-伍华林

ISO 26262



1. Vocabulary

2. Management of functional safety
- 2-5 Overall safety management
- 2-6 Project dependent safety management
- 2-7 Safety management regarding production, operation, service and decommissioning

3. Concept phase
- 3-5 Item definition
- 3-6 Hazard analysis and risk assessment
- 3-7 Functional safety concept

4. Product development at the system level
- 4-5 General topics for the product development at the system level
- 4-6 Technical safety concept
- 4-7 System and item integration and testing
- 4-8 Safety validation

7. Production, operation, service and decommissioning
- 7-5 Planning for production, operation, service and decommissioning
- 7-6 Production
- 7-7 Operation, service and decommissioning

12. Adaptation of ISO 26262 for motorcycles
- 12-5 General topics for adaptation for motorcycles
- 12-6 Safety culture
- 12-7 Confirmation measures
- 12-8 Hazard analysis and risk assessment
- 12-9 Vehicle integration and testing
- 12-10 Safety validation

5. Product development at the hardware level
- 5-5 General topics for the product development at the hardware level
- 5-6 Specification of hardware safety requirements
- 5-7 Hardware design
- 5-8 Evaluation of the hardware architectural metrics
- 5-9 Evaluation of safety goal violations due to random hardware failures
- 5-10 Hardware integration and verification

6. Product development at the software level
- 6-5 General topics for the product development at the software level
- 6-6 Specification of software safety requirements
- 6-7 Software archtectural design
- 6-8 Software unit design and implementation
- 6-9 Software unit verification
- 6-10 Software integration and verification
- 6-11 Testing of the embedded software

8. Supporting processes
- 8-5 Interfaces within distributed developments
- 8-6 Specification and management of safety requirements
- 8-7 Configuration management
- 8-8 Change management
- 8-9 Verification
- 8-10 Documentation management
- 8-11 Confidence in the use of software tools
- 8-12 Qualification of software components
- 8-13 Evaluation of hardware elements
- 8-14 Proven in use argument
- 8-15 Interfacing an application that is out of scope of ISO 26262
- 8-16 Integration of safety-related systems not developed according to ISO 26262

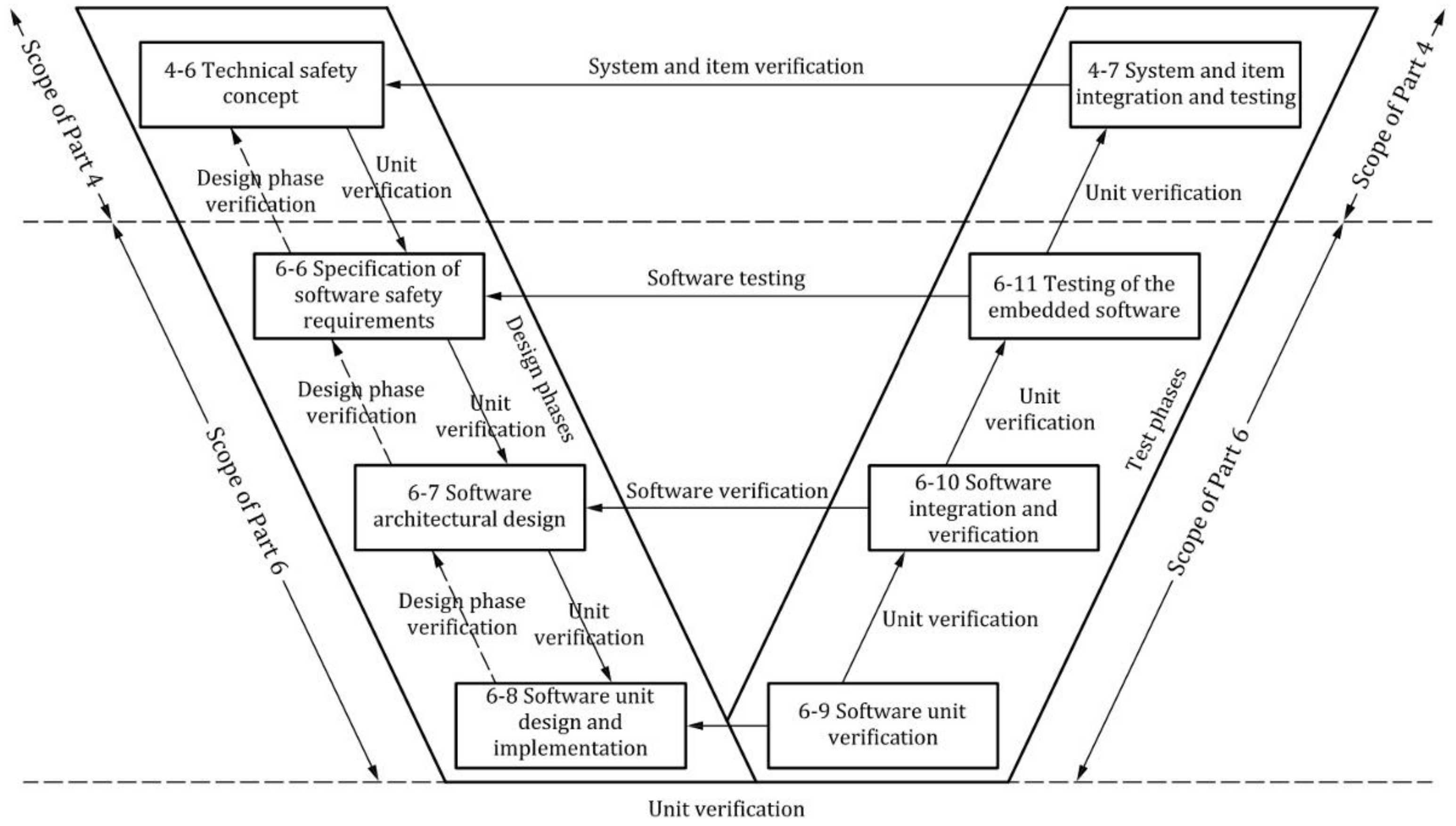
9. Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses
- 9-5 Requirements decomposition with respect to ASIL tailoring
- 9-6 Criteria for coexistence of elements
- 9-7 Analysis of dependent failures
- 9-8 Safety analyses

10. Guidelines on ISO 26262

11. Guidelines on application of ISO 26262 to semiconductors

Figure 1 — Overview of the ISO 26262 series of standards

ISO 26262-6



Scope of Part 4
4-6 Technical safety concept
System and item verification
4-7 System and item integration and testing
Scope of Part 4
Design phase verification
Unit verification
Unit verification
6-6 Specification of software safety requirements
Software testing
6-11 Testing of the embedded software
Design phases
Design phase verification
Unit verification
Unit verification
Test phases
Scope of Part 6
Scope of Part 6
6-7 Software architectural design
Software verification
6-10 Software integration and verification
Design phase verification
Unit verification
Unit verification
6-8 Software unit design and implementation
6-9 Software unit verification
Unit verification

NOTE Within the figure, the specific clauses of each part of the ISO 26262 series of standards are indicated in the following manner: “m-n”, where “m” represents the number of the part and “n” indicates the number of the clause, e.g. “4-7” represents ISO 26262-4:2018, Clause 7.

Figure 2 — Reference phase model for the product development at the software level

Approaches to ASIL-D Software Compliance

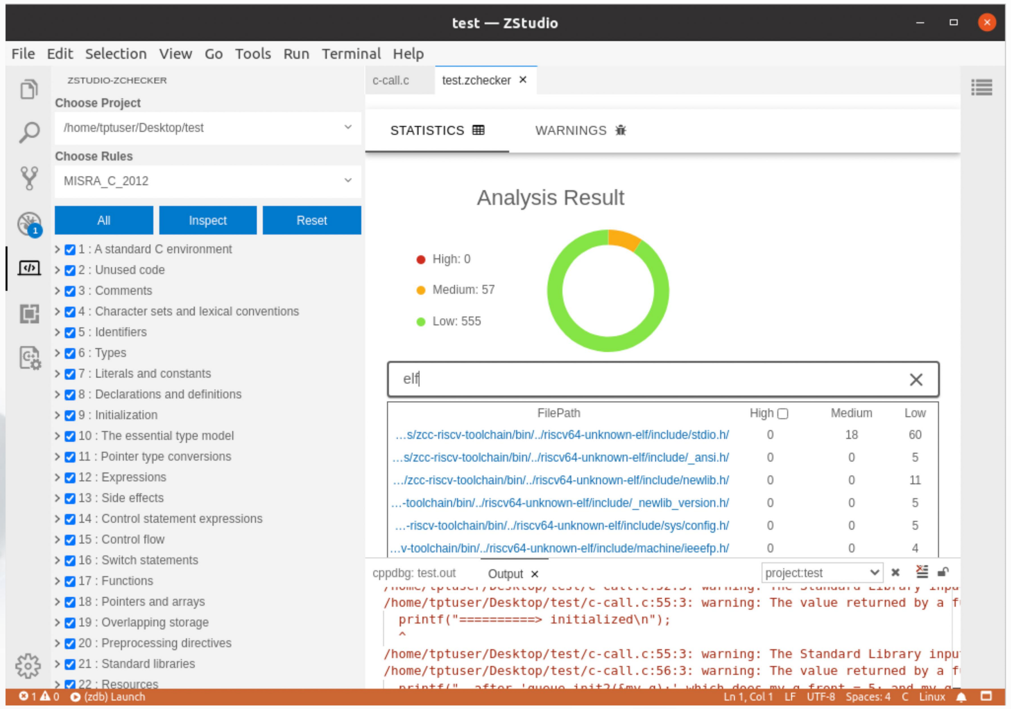
- Linter & AST Matcher
- Abstract Interpretation
- Symbolic Execution
- One-and-a-half-order Theorem Prover
- Program Coverage Analysis
- Testcase Auto-generation
- Library Modeling
- ……

Topics for next time

Those approaches can also be applied to HDL functional safety check

ASIL-D Compliance Software Enabler - zchecker

Supports over 150 MISRA C 2012 rules
Supports HIS code complexity check
Supports for code coverage and testcase auto generation is on the way
Supports for MISRA C++ 202x (AutoSAR C++) is on the way



Choose Project: /home/tptuser/Desktop/test

Choose Rules: MISRA_C_2012

- ✓ 1 : A standard C environment
- ✓ 2 : Unused code
- ✓ 3 : Comments
- ✓ 4 : Character sets and lexical conventions
- ✓ 5 : Identifiers
- ✓ 6 : Types
- ✓ 7 : Literals and constants
- ✓ 8 : Declarations and definitions
- ✓ 9 : Initialization
- ✓ 10 : The essential type model
- ✓ 11 : Pointer type conversions
- ✓ 12 : Expressions
- ✓ 13 : Side effects
- ✓ 14 : Control statement expressions
- ✓ 15 : Control flow
- ✓ 16 : Switch statements
- ✓ 17 : Functions
- ✓ 18 : Pointers and arrays
- ✓ 19 : Overlapping storage
- ✓ 20 : Preprocessing directives
- ✓ 21 : Standard libraries
- ✓ 22 : Resources

Analysis Result

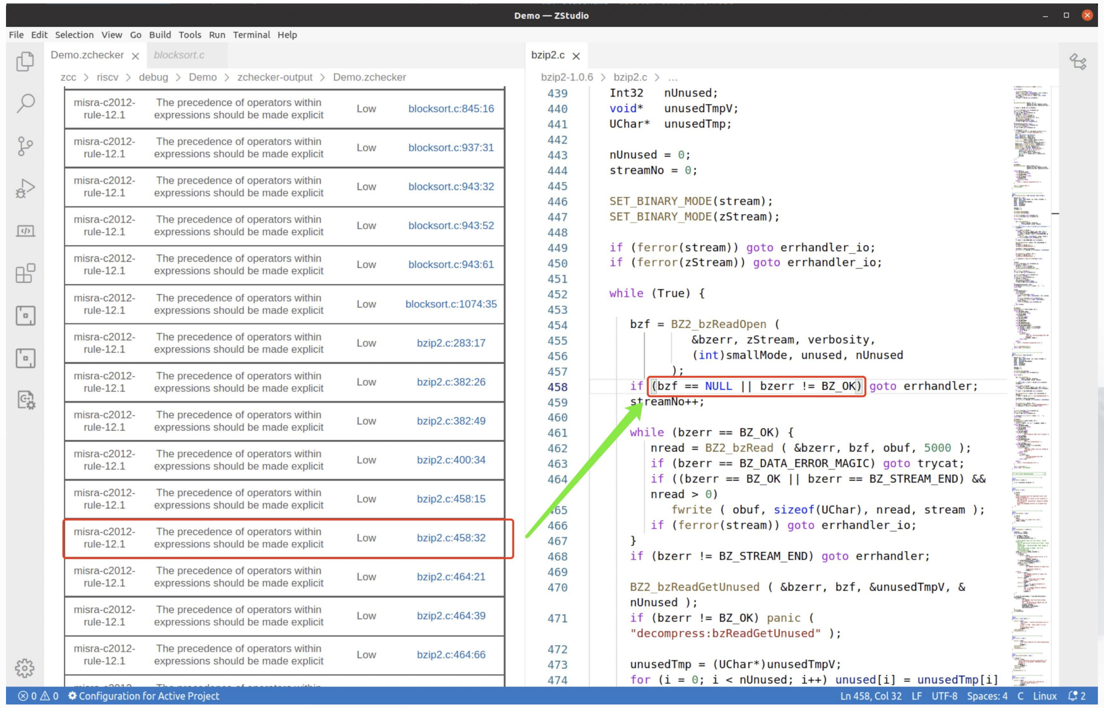
- ● High: 0
- ● Medium: 57
- ● Low: 555

elf

FilePath	High	Medium	Low
…s/zcc-riscv-toolchain/bin/../riscv64-unknown-elf/include/stdio.h/	0	18	60
…s/zcc-riscv-toolchain/bin/../riscv64-unknown-elf/include/_ansi.h/	0	0	5
…/zcc-riscv-toolchain/bin/../riscv64-unknown-elf/include/newlib.h/	0	0	11
…-toolchain/bin/../riscv64-unknown-elf/include/_newlib_version.h/	0	0	5
…-riscv-toolchain/bin/../riscv64-unknown-elf/include/sys/config.h/	0	0	5
…v-toolchain/bin/../riscv64-unknown-elf/include/machine/ieeefp.h/	0	0	4

project:test

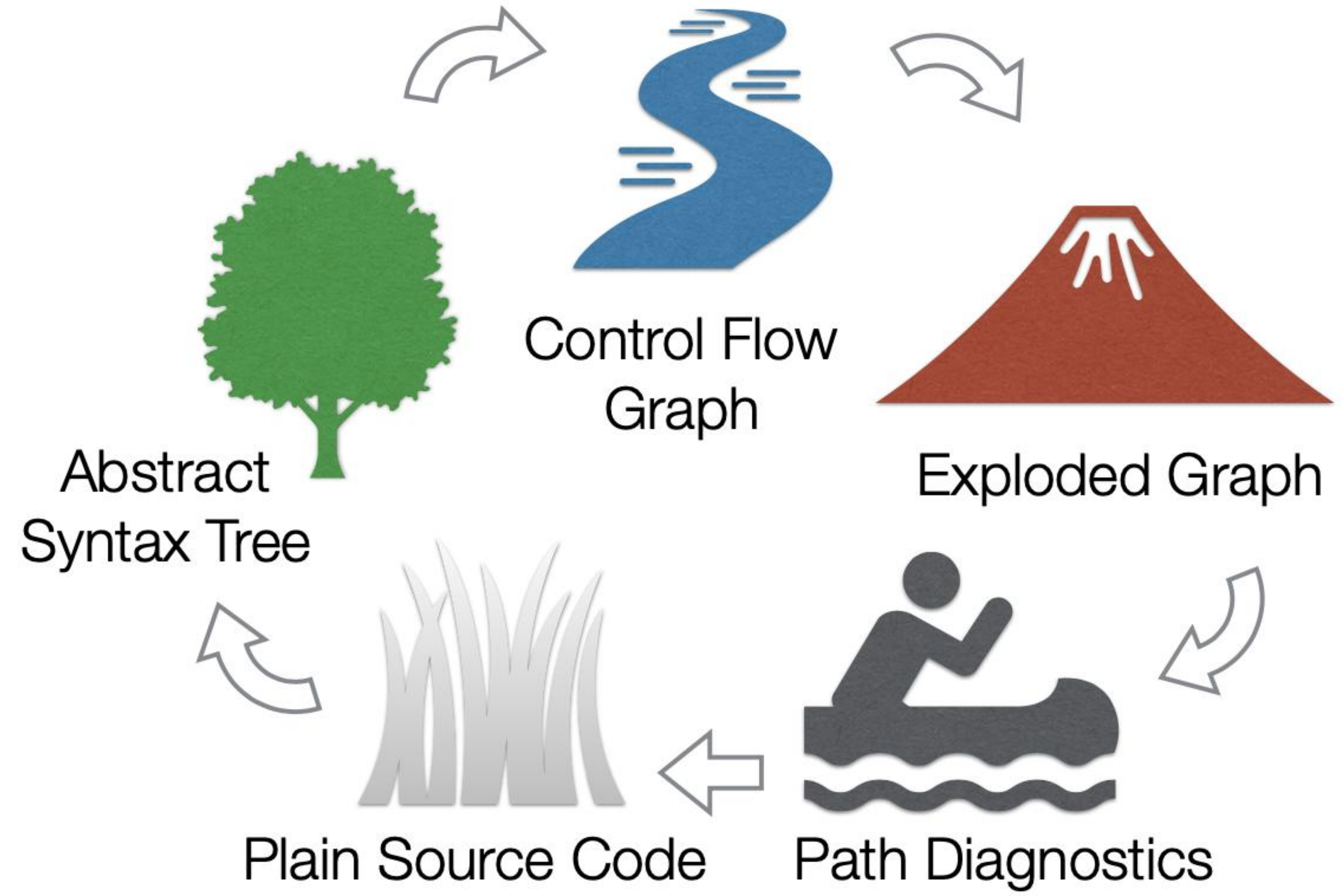
```
/home/tptuser/Desktop/test/c-call.c:55:3: warning: The value returned by a f
  printf("==========> initialized\n");
  ^
/home/tptuser/Desktop/test/c-call.c:55:3: warning: The Standard Library inpu
/home/tptuser/Desktop/test/c-call.c:56:3: warning: The value returned by a f
```


Demo — ZStudio
File Edit Selection View Go Build Tools Run Terminal Help
Demo.zchecker ×
blocksort.c
bzip2.c ×
zcc > riscv > debug > Demo > zchecker-output > Demo.zchecker
bzip2-1.0.6 > bzip2.c > …
Configuration for Active Project
Ln 458, Col 32 LF UTF-8 Spaces: 4 C Linux

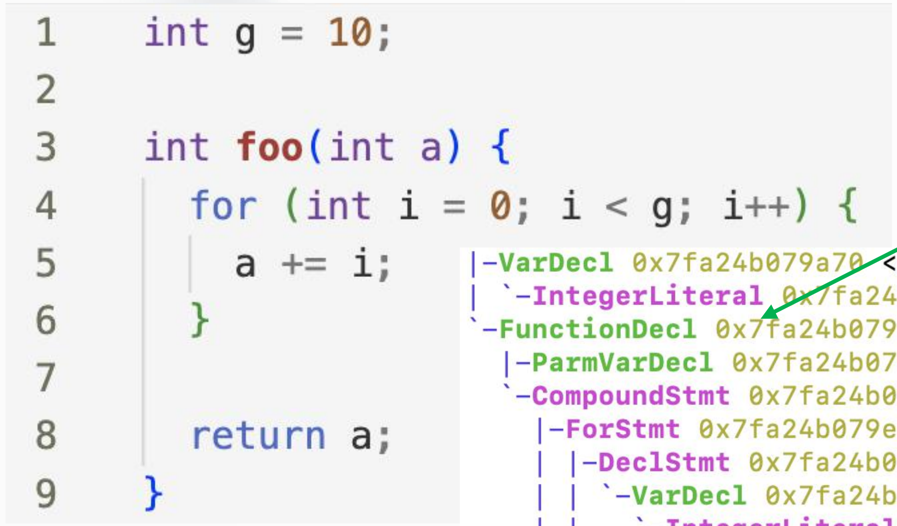
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	blocksort.c:845:16
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	blocksort.c:937:31
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	blocksort.c:943:32
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	blocksort.c:943:52
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	blocksort.c:943:61
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	blocksort.c:1074:35
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:283:17
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:382:26
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:382:49
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:400:34
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:458:15
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:458:32
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:464:21
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:464:39
misra-c2012-rule-12.1	The precedence of operators within expressions should be made explicit	Low	bzip2.c:464:66

```c
   Int32   nUnused;
   void*   unusedTmpV;
   UChar*  unusedTmp;

   nUnused = 0;
   streamNo = 0;

   SET_BINARY_MODE(stream);
   SET_BINARY_MODE(zStream);

   if (ferror(stream)) goto errhandler_io;
   if (ferror(zStream)) goto errhandler_io;

   while (True) {

      bzf = BZ2_bzReadOpen (
               &bzerr, zStream, verbosity,
               (int)smallMode, unused, nUnused
            );
      if (bzf == NULL || bzerr != BZ_OK) goto errhandler;
      streamNo++;

      while (bzerr == BZ_OK) {
         nread = BZ2_bzRead ( &bzerr, bzf, obuf, 5000 );
         if (bzerr == BZ_DATA_ERROR_MAGIC) goto trycat;
         if ((bzerr == BZ_OK || bzerr == BZ_STREAM_END) &&
         nread > 0)
            fwrite ( obuf, sizeof(UChar), nread, stream );
         if (ferror(stream)) goto errhandler_io;
      }
      if (bzerr != BZ_STREAM_END) goto errhandler;

      BZ2_bzReadGetUnused ( &bzerr, bzf, &unusedTmpV, &
      nUnused );
      if (bzerr != BZ_OK) panic (
      "decompress:bzReadGetUnused" );

      unusedTmp = (UChar*)unusedTmpV;
      for (i = 0; i < nUnused; i++) unused[i] = unusedTmp[i]
```

Algorithms and Data Structure in Clang Static Analyzer



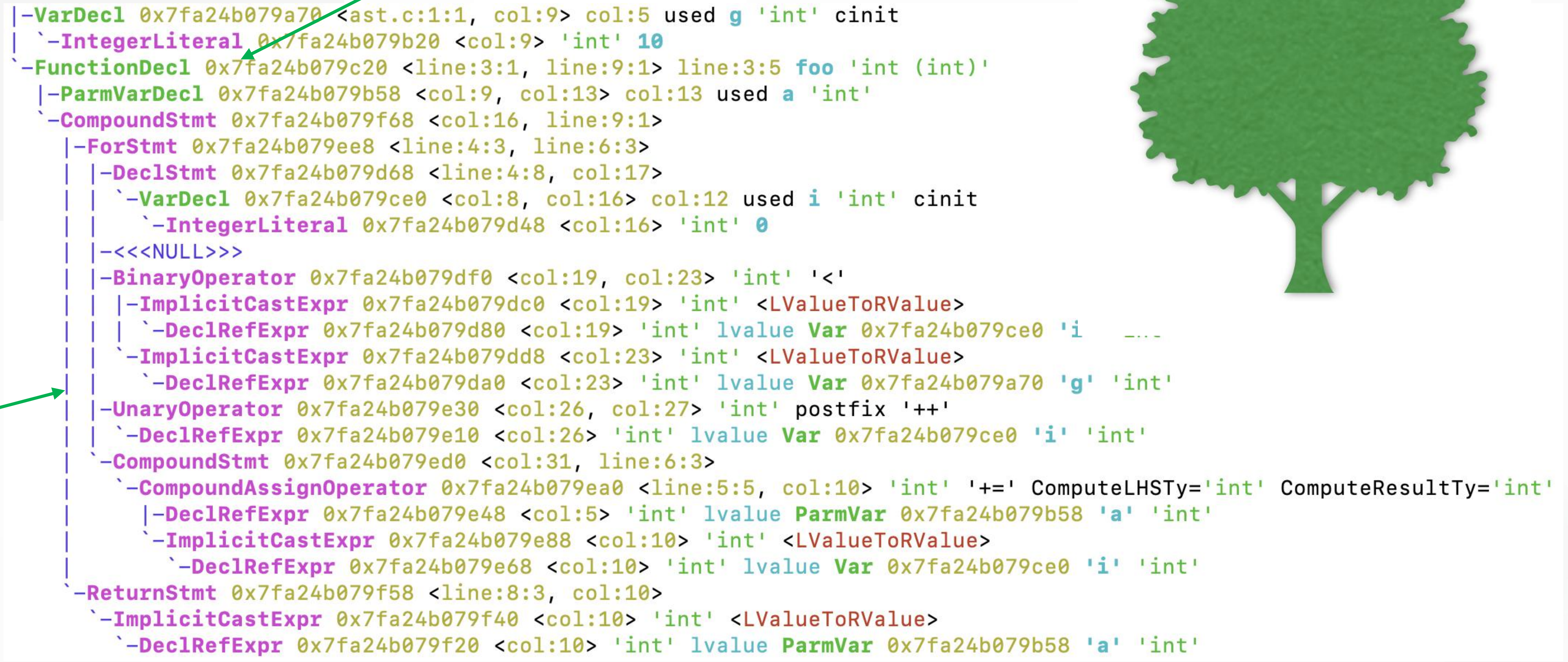
AST: How Compiler Sees Your Code



```c
int g = 10;

int foo(int a) {
  for (int i = 0; i < g; i++) {
    a += i;
  }

  return a;
}
```

Node

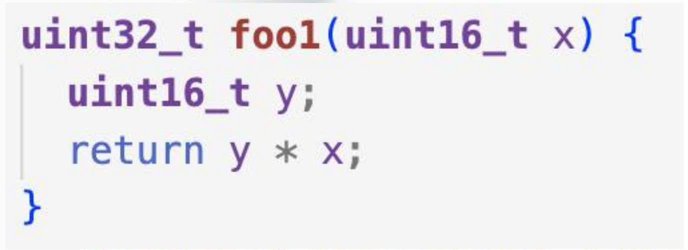
Edge



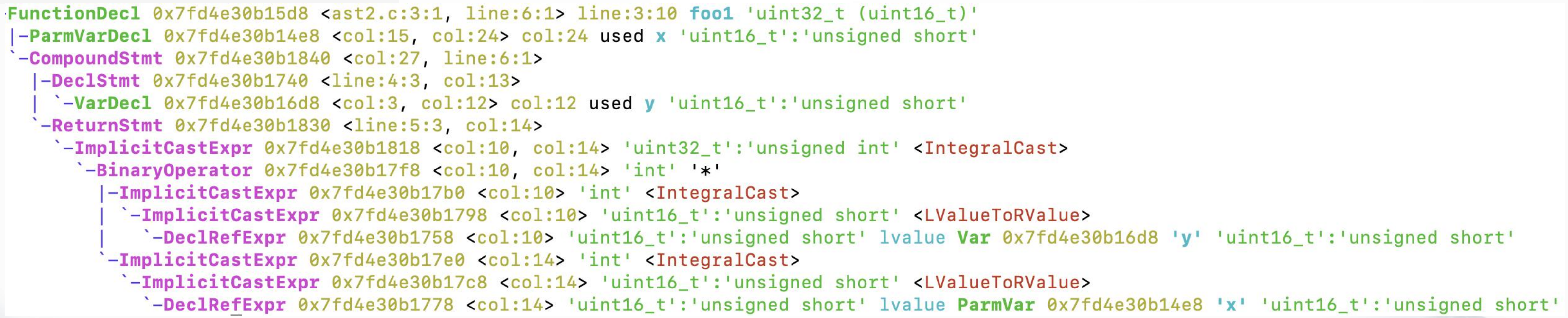
```
|-VarDecl 0x7fa24b079a70 <ast.c:1:1, col:9> col:5 used g 'int' cinit
| `-IntegerLiteral 0x7fa24b079b20 <col:9> 'int' 10
`-FunctionDecl 0x7fa24b079c20 <line:3:1, line:9:1> line:3:5 foo 'int (int)'
  |-ParmVarDecl 0x7fa24b079b58 <col:9, col:13> col:13 used a 'int'
  `-CompoundStmt 0x7fa24b079f68 <col:16, line:9:1>
    |-ForStmt 0x7fa24b079ee8 <line:4:3, line:6:3>
    | |-DeclStmt 0x7fa24b079d68 <line:4:8, col:17>
    | | `-VarDecl 0x7fa24b079ce0 <col:8, col:16> col:12 used i 'int' cinit
    | |   `-IntegerLiteral 0x7fa24b079d48 <col:16> 'int' 0
    | |-<<<NULL>>>
    | |-BinaryOperator 0x7fa24b079df0 <col:19, col:23> 'int' '<'
    | | |-ImplicitCastExpr 0x7fa24b079dc0 <col:19> 'int' <LValueToRValue>
    | | | `-DeclRefExpr 0x7fa24b079d80 <col:19> 'int' lvalue Var 0x7fa24b079ce0 'i
    | | `-ImplicitCastExpr 0x7fa24b079dd8 <col:23> 'int' <LValueToRValue>
    | |   `-DeclRefExpr 0x7fa24b079da0 <col:23> 'int' lvalue Var 0x7fa24b079a70 'g' 'int'
    | |-UnaryOperator 0x7fa24b079e30 <col:26, col:27> 'int' postfix '++'
    | | `-DeclRefExpr 0x7fa24b079e10 <col:26> 'int' lvalue Var 0x7fa24b079ce0 'i' 'int'
    | `-CompoundStmt 0x7fa24b079ed0 <col:31, line:6:3>
    |   `-CompoundAssignOperator 0x7fa24b079ea0 <line:5:5, col:10> 'int' '+=' ComputeLHSTy='int' ComputeResultTy='int'
    |     |-DeclRefExpr 0x7fa24b079e48 <col:5> 'int' lvalue ParmVar 0x7fa24b079b58 'a' 'int'
    |     `-ImplicitCastExpr 0x7fa24b079e88 <col:10> 'int' <LValueToRValue>
    |       `-DeclRefExpr 0x7fa24b079e68 <col:10> 'int' lvalue Var 0x7fa24b079ce0 'i' 'int'
    `-ReturnStmt 0x7fa24b079f58 <line:8:3, col:10>
      `-ImplicitCastExpr 0x7fa24b079f40 <col:10> 'int' <LValueToRValue>
        `-DeclRefExpr 0x7fa24b079f20 <col:10> 'int' lvalue ParmVar 0x7fa24b079b58 'a' 'int'
```

`clang -S -Xclang -ast-dump ast.c`

AST Matcher: Pattern Matches on AST



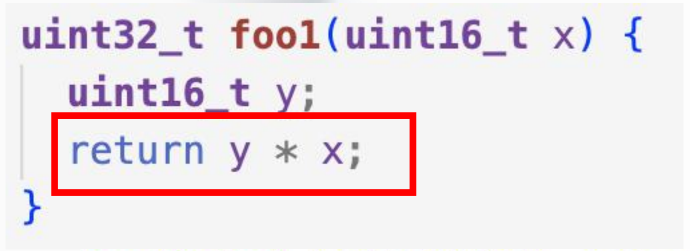
```
uint32_t foo1(uint16_t x) {
  uint16_t y;
  return y * x;
}
```



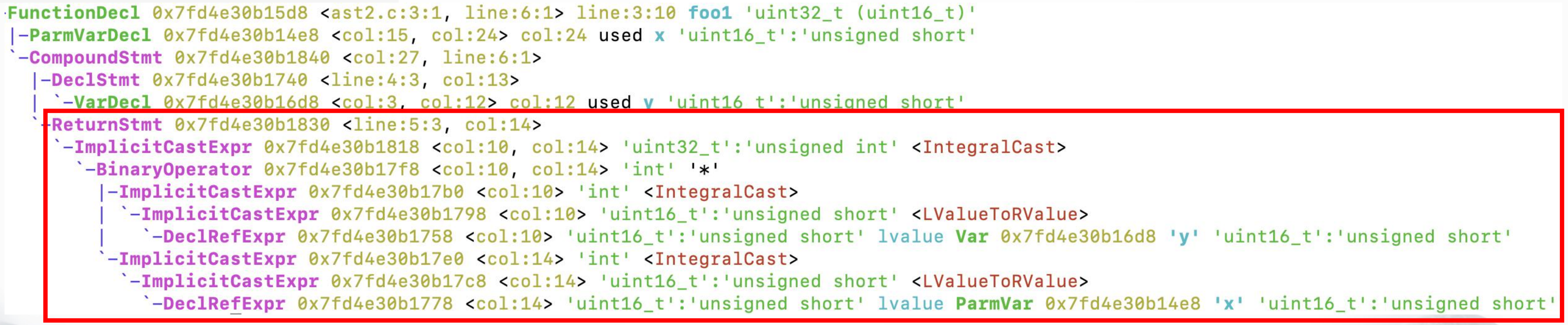
```
FunctionDecl 0x7fd4e30b15d8 <ast2.c:3:1, line:6:1> line:3:10 foo1 'uint32_t (uint16_t)'
|-ParmVarDecl 0x7fd4e30b14e8 <col:15, col:24> col:24 used x 'uint16_t':'unsigned short'
`-CompoundStmt 0x7fd4e30b1840 <col:27, line:6:1>
  |-DeclStmt 0x7fd4e30b1740 <line:4:3, col:13>
  | `-VarDecl 0x7fd4e30b16d8 <col:3, col:12> col:12 used y 'uint16_t':'unsigned short'
  `-ReturnStmt 0x7fd4e30b1830 <line:5:3, col:14>
    `-ImplicitCastExpr 0x7fd4e30b1818 <col:10, col:14> 'uint32_t':'unsigned int' <IntegralCast>
      `-BinaryOperator 0x7fd4e30b17f8 <col:10, col:14> 'int' '*'
        |-ImplicitCastExpr 0x7fd4e30b17b0 <col:10> 'int' <IntegralCast>
        | `-ImplicitCastExpr 0x7fd4e30b1798 <col:10> 'uint16_t':'unsigned short' <LValueToRValue>
        |   `-DeclRefExpr 0x7fd4e30b1758 <col:10> 'uint16_t':'unsigned short' lvalue Var 0x7fd4e30b16d8 'y' 'uint16_t':'unsigned short'
        `-ImplicitCastExpr 0x7fd4e30b17e0 <col:14> 'int' <IntegralCast>
          `-ImplicitCastExpr 0x7fd4e30b17c8 <col:14> 'uint16_t':'unsigned short' <LValueToRValue>
            `-DeclRefExpr 0x7fd4e30b1778 <col:14> 'uint16_t':'unsigned short' lvalue ParmVar 0x7fd4e30b14e8 'x' 'uint16_t':'unsigned short'
```

MISRA C2012 Rule 10.6[Required]: The value of a composite expression shall not be assigned to an object with wider essential type

AST Matcher: Pattern Matches on AST



```
uint32_t foo1(uint16_t x) {
  uint16_t y;
  return y * x;
}
```

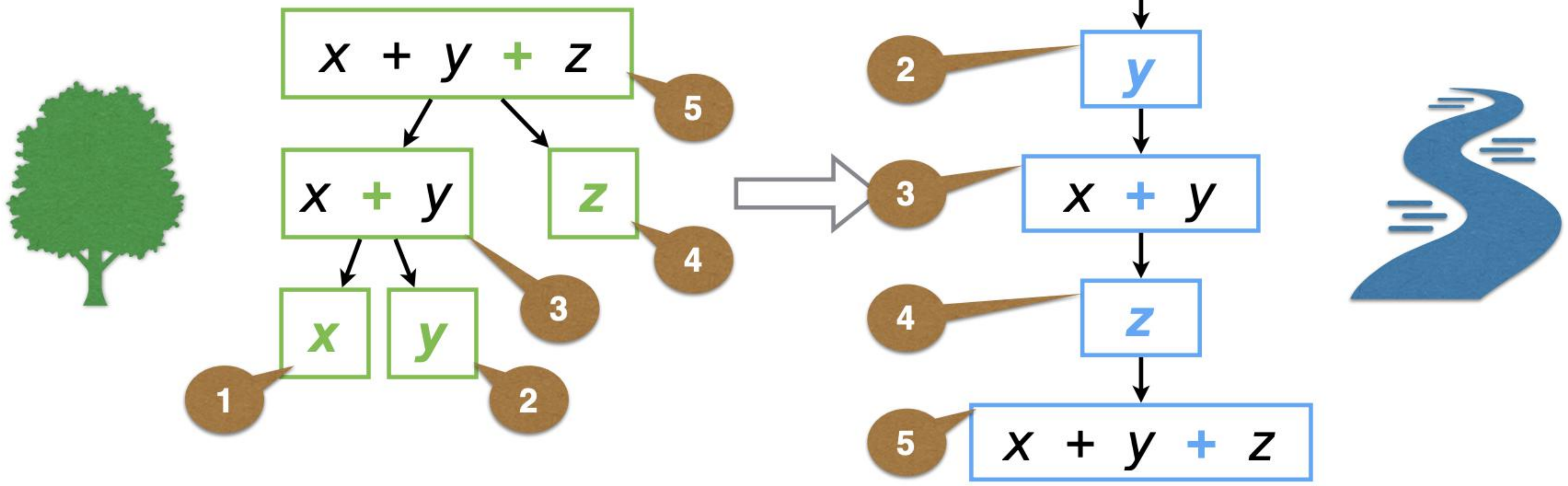


```
FunctionDecl 0x7fd4e30b15d8 <ast2.c:3:1, line:6:1> line:3:10 foo1 'uint32_t (uint16_t)'
|-ParmVarDecl 0x7fd4e30b14e8 <col:15, col:24> col:24 used x 'uint16_t':'unsigned short'
`-CompoundStmt 0x7fd4e30b1840 <col:27, line:6:1>
  |-DeclStmt 0x7fd4e30b1740 <line:4:3, col:13>
  | `-VarDecl 0x7fd4e30b16d8 <col:3, col:12> col:12 used y 'uint16_t':'unsigned short'
  `-ReturnStmt 0x7fd4e30b1830 <line:5:3, col:14>
    `-ImplicitCastExpr 0x7fd4e30b1818 <col:10, col:14> 'uint32_t':'unsigned int' <IntegralCast>
      `-BinaryOperator 0x7fd4e30b17f8 <col:10, col:14> 'int' '*'
        |-ImplicitCastExpr 0x7fd4e30b17b0 <col:10> 'int' <IntegralCast>
        | `-ImplicitCastExpr 0x7fd4e30b1798 <col:10> 'uint16_t':'unsigned short' <LValueToRValue>
        |   `-DeclRefExpr 0x7fd4e30b1758 <col:10> 'uint16_t':'unsigned short' lvalue Var 0x7fd4e30b16d8 'y' 'uint16_t':'unsigned short'
        `-ImplicitCastExpr 0x7fd4e30b17e0 <col:14> 'int' <IntegralCast>
          `-ImplicitCastExpr 0x7fd4e30b17c8 <col:14> 'uint16_t':'unsigned short' <LValueToRValue>
            `-DeclRefExpr 0x7fd4e30b1778 <col:14> 'uint16_t':'unsigned short' lvalue ParmVar 0x7fd4e30b14e8 'x' 'uint16_t':'unsigned short'
```

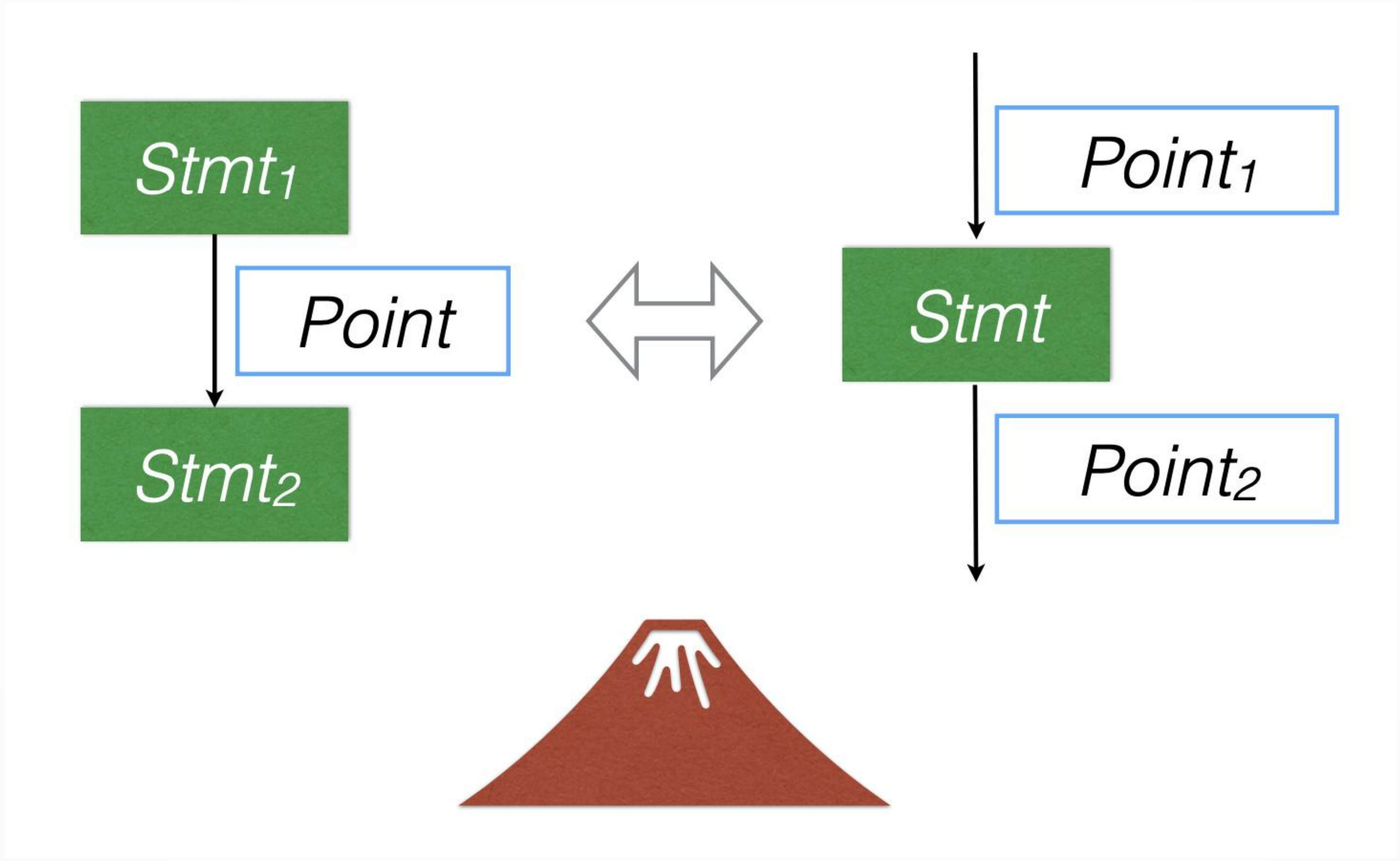
MISRA C2012 Rule 10.6[Required]: The value of a composite expression shall not be assigned to an object with wider essential type

CFG: Order in which Statements are Executed

- *Nodes*: usually AST statements
- *Edges*: “executed-after” relation

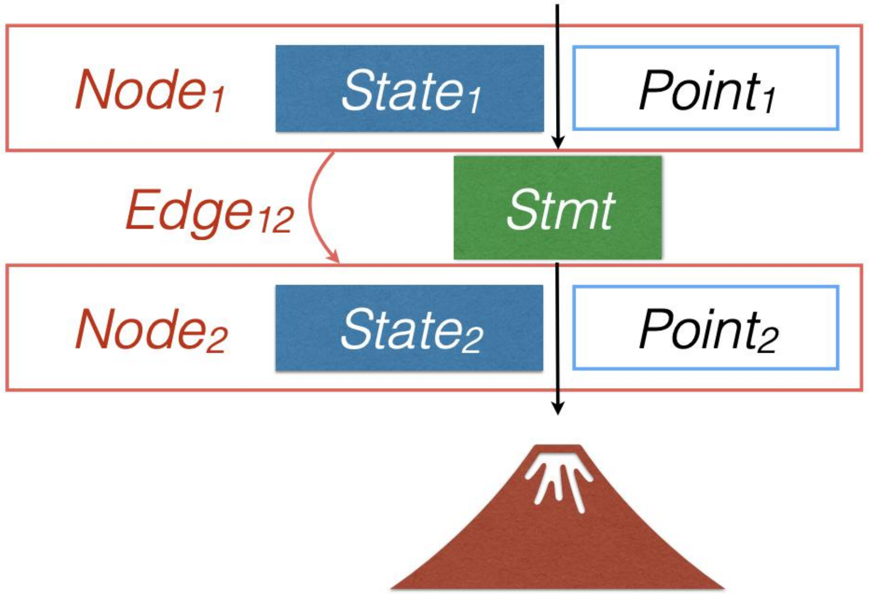


Program Points

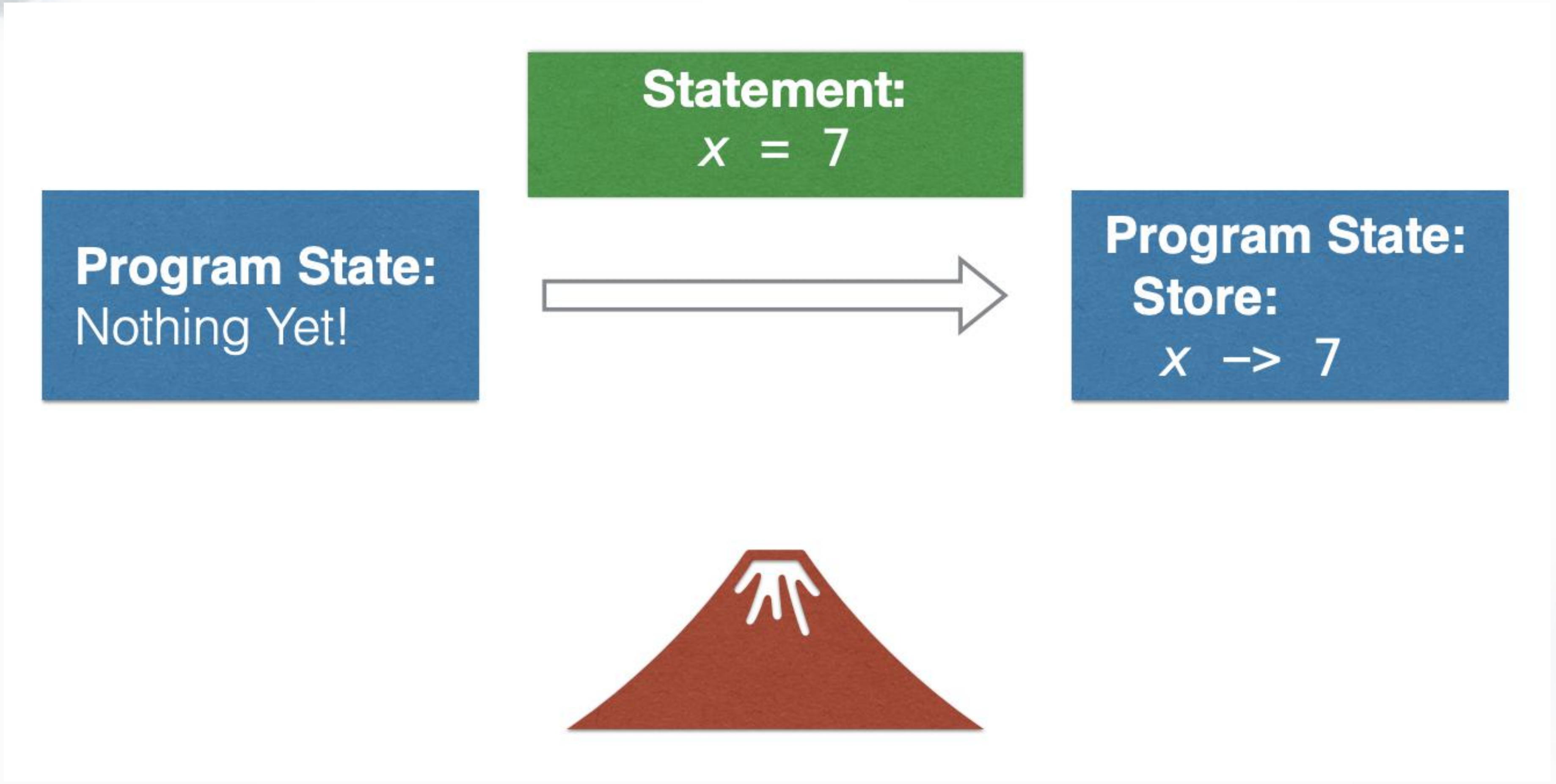


Exploded Graph: Paths through CFG

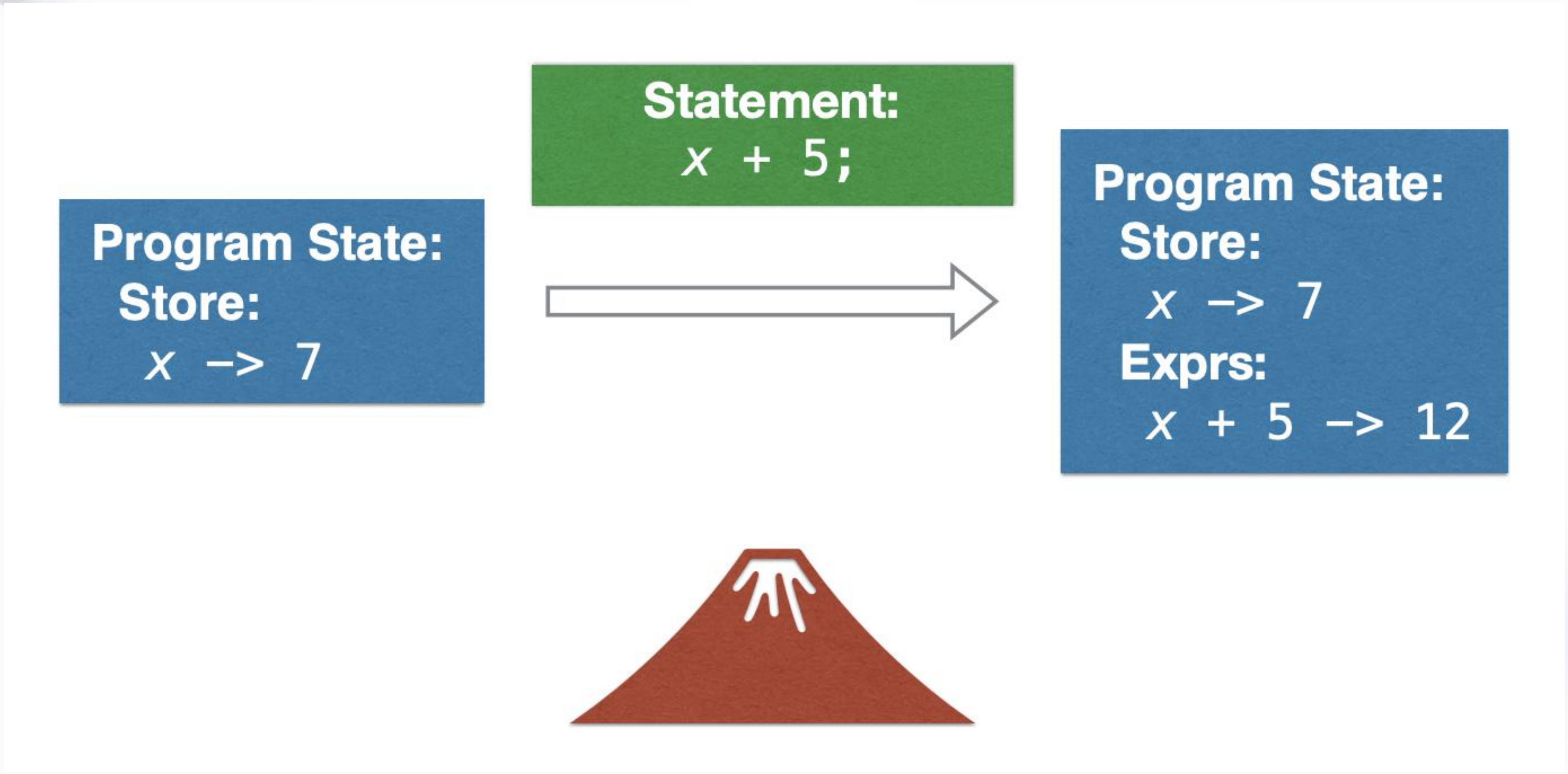
- Nodes： (Point， State) pairs
  - Program Point： A point between statements
  - Program State： A record of effects of statements evaluated so far
- Edges： An edge from (Point$_1$， State$_1$) to (Point$_2$， State$_2$) ＝ Statement *between* Point$_1$ and Points$_2$ *updates* State$_1$ to State$_2$



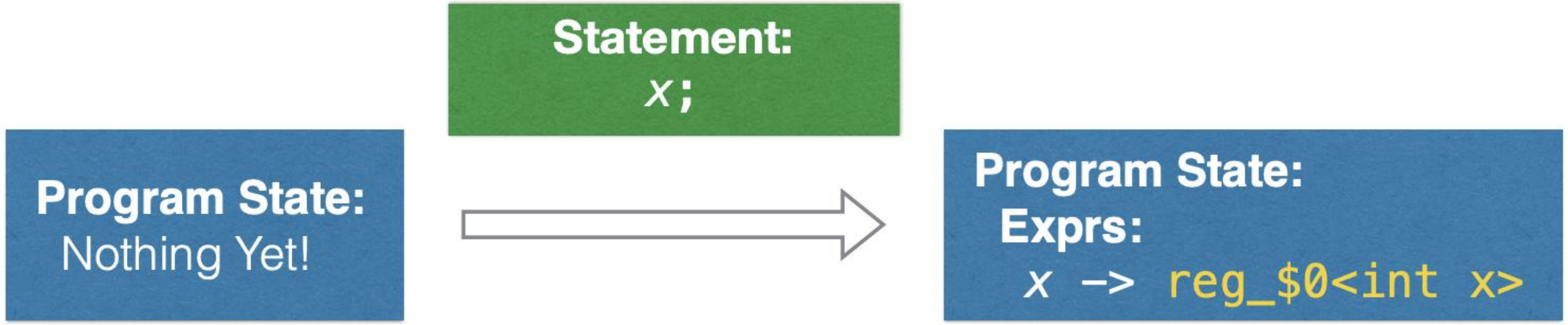
Effects of Assignments: Store



Values of Expressions

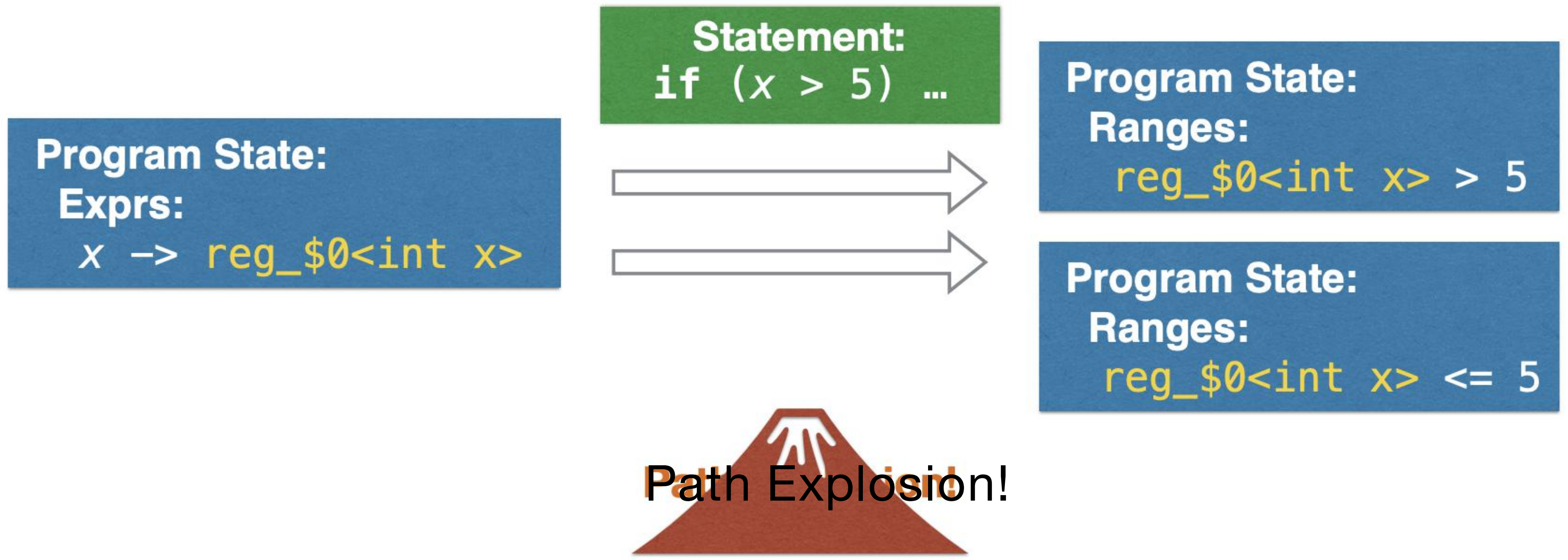


What if it’ s not in the store?

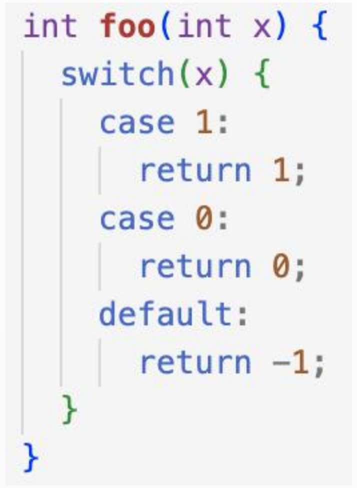


```
// example:
int foo(int x) {
  return x;
}
```

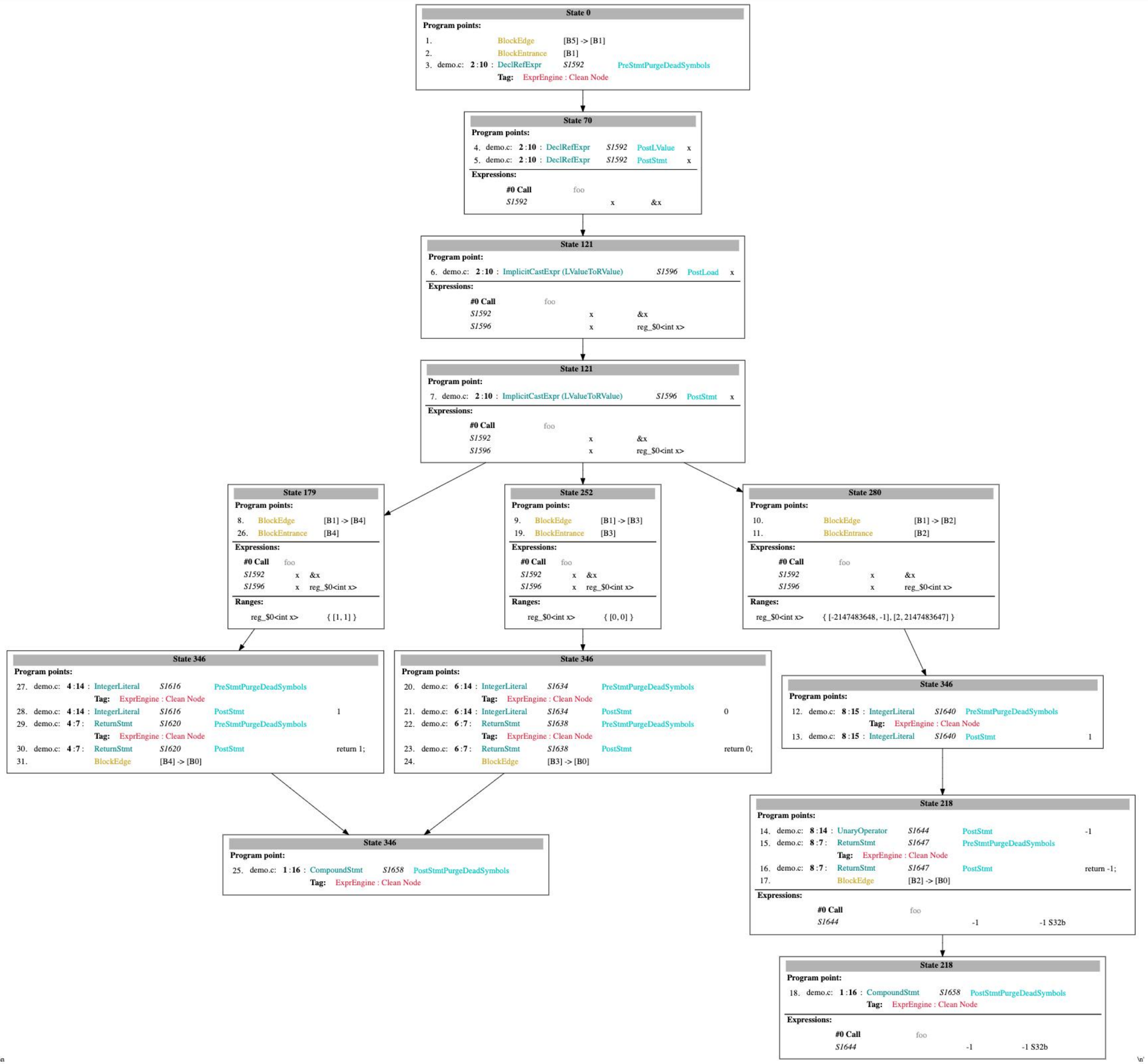
Effects of Branches: Constraints



Exploded Graph in Real World



```c
int foo(int x) {
  switch(x) {
    case 1:
      return 1;
    case 0:
      return 0;
    default:
      return -1;
  }
}
```



```
State 0
Program points:
 1.                  BlockEdge      [B5] -> [B1]
 2.                  BlockEntrance  [B1]
 3. demo.c:  2 :10 : DeclRefExpr    S1592   PreStmtPurgeDeadSymbols
                     Tag:   ExprEngine : Clean Node

State 70
Program points:
 4. demo.c:  2 :10 : DeclRefExpr    S1592   PostLValue   x
 5. demo.c:  2 :10 : DeclRefExpr    S1592   PostStmt     x
Expressions:
      #0 Call       foo
      S1592                   x       &x

State 121
Program point:
 6. demo.c:  2 :10 : ImplicitCastExpr (LValueToRValue)   S1596   PostLoad   x
Expressions:
      #0 Call       foo
      S1592                   x       &x
      S1596                   x       reg_$0<int x>

State 121
Program point:
 7. demo.c:  2 :10 : ImplicitCastExpr (LValueToRValue)   S1596   PostStmt   x
Expressions:
      #0 Call       foo
      S1592                   x       &x
      S1596                   x       reg_$0<int x>

State 179
Program points:
 8.   BlockEdge       [B1] -> [B4]
 26.  BlockEntrance   [B4]
Expressions:
  #0 Call   foo
  S1592     x   &x
  S1596     x   reg_$0<int x>
Ranges:
  reg_$0<int x>     { [1, 1] }

State 252
Program points:
 9.   BlockEdge       [B1] -> [B3]
 19.  BlockEntrance   [B3]
Expressions:
  #0 Call   foo
  S1592     x   &x
  S1596     x   reg_$0<int x>
Ranges:
  reg_$0<int x>     { [0, 0] }

State 280
Program points:
 10.                 BlockEdge       [B1] -> [B2]
 11.                 BlockEntrance   [B2]
Expressions:
      #0 Call   foo
      S1592               x       &x
      S1596               x       reg_$0<int x>
Ranges:
 reg_$0<int x>    { [-2147483648, -1], [2, 2147483647] }

State 346
Program points:
 27. demo.c:  4 :14 : IntegerLiteral   S1616   PreStmtPurgeDeadSymbols
                      Tag:  ExprEngine : Clean Node
 28. demo.c:  4 :14 : IntegerLiteral   S1616   PostStmt                  1
 29. demo.c:  4 :7 :  ReturnStmt       S1620   PreStmtPurgeDeadSymbols
                      Tag:  ExprEngine : Clean Node
 30. demo.c:  4 :7 :  ReturnStmt       S1620   PostStmt                  return 1;
 31.                  BlockEdge        [B4] -> [B0]

State 346
Program points:
 20. demo.c:  6 :14 : IntegerLiteral   S1634   PreStmtPurgeDeadSymbols
                      Tag:  ExprEngine : Clean Node
 21. demo.c:  6 :14 : IntegerLiteral   S1634   PostStmt                  0
 22. demo.c:  6 :7 :  ReturnStmt       S1638   PreStmtPurgeDeadSymbols
                      Tag:  ExprEngine : Clean Node
 23. demo.c:  6 :7 :  ReturnStmt       S1638   PostStmt                  return 0;
 24.                  BlockEdge        [B3] -> [B0]

State 346
Program points:
 12. demo.c:  8 :15 : IntegerLiteral   S1640   PreStmtPurgeDeadSymbols
                      Tag:  ExprEngine : Clean Node
 13. demo.c:  8 :15 : IntegerLiteral   S1640   PostStmt                  1

State 346
Program point:
 25. demo.c:  1 :16 : CompoundStmt   S1658   PostStmtPurgeDeadSymbols
                      Tag:  ExprEngine : Clean Node

State 218
Program points:
 14. demo.c:  8 :14 : UnaryOperator   S1644   PostStmt                  -1
 15. demo.c:  8 :7 :  ReturnStmt      S1647   PreStmtPurgeDeadSymbols
                      Tag:  ExprEngine : Clean Node
 16. demo.c:  8 :7 :  ReturnStmt      S1647   PostStmt                  return -1;
 17.                  BlockEdge       [B2] -> [B0]
Expressions:
      #0 Call        foo
      S1644                     -1          -1 S32b

State 218
Program point:
 18. demo.c:  1 :16 : CompoundStmt   S1658   PostStmtPurgeDeadSymbols
                      Tag:  ExprEngine : Clean Node
Expressions:
      #0 Call        foo
      S1644                     -1          -1 S32b
```

提供专业的工具和服务帮助客户软件达成ASIL-D

THANKS

兆松科技联合创始人&CTO 伍华林
电话/微信：18217640818

全球开源技术峰会
THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE