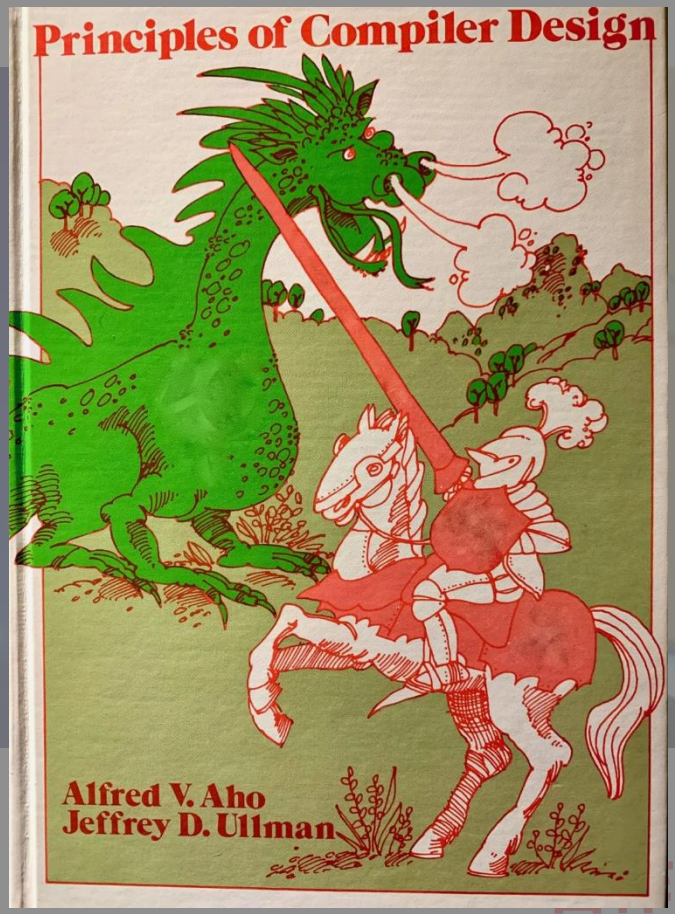
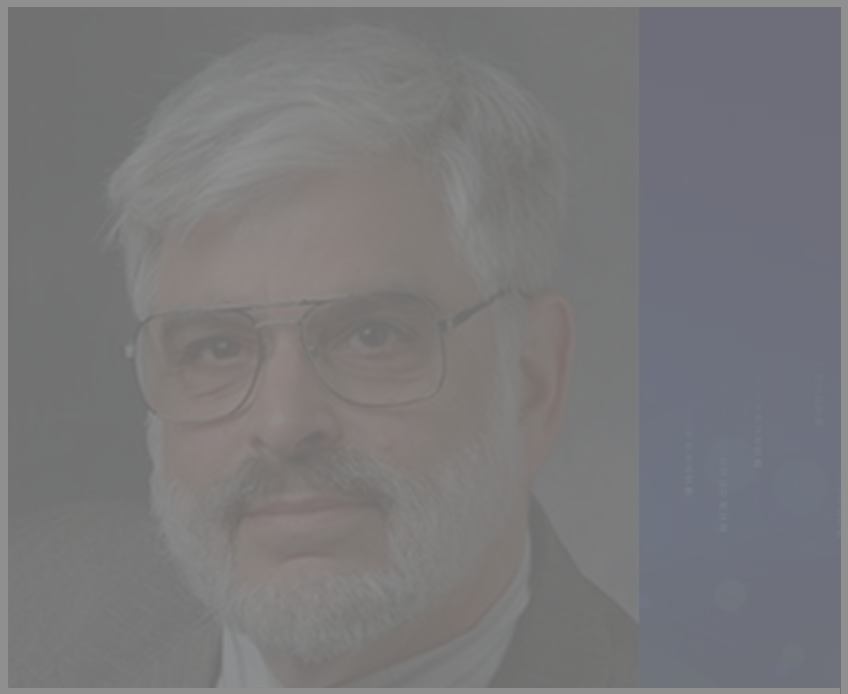
编译器测试研究进展

大连理工大学　江 贺

合作几十年的老友。


Principles of Compiler Design
Alfred V. Aho
Jeffrey D. Ullman



几十年的老友。

龙书


2020年图灵奖得主
Alfred Aho
Jeffrey Ullman

图灵奖-计算机界最高奖

———从1966年至2020年，图灵奖共授予74名获奖者，约1/3与编程语言、编译器有关

第一款编译器

John Backus（1924.12-2007.3）
1977年图灵奖获得者
Fortran 语言之父

- **第一个编译器**

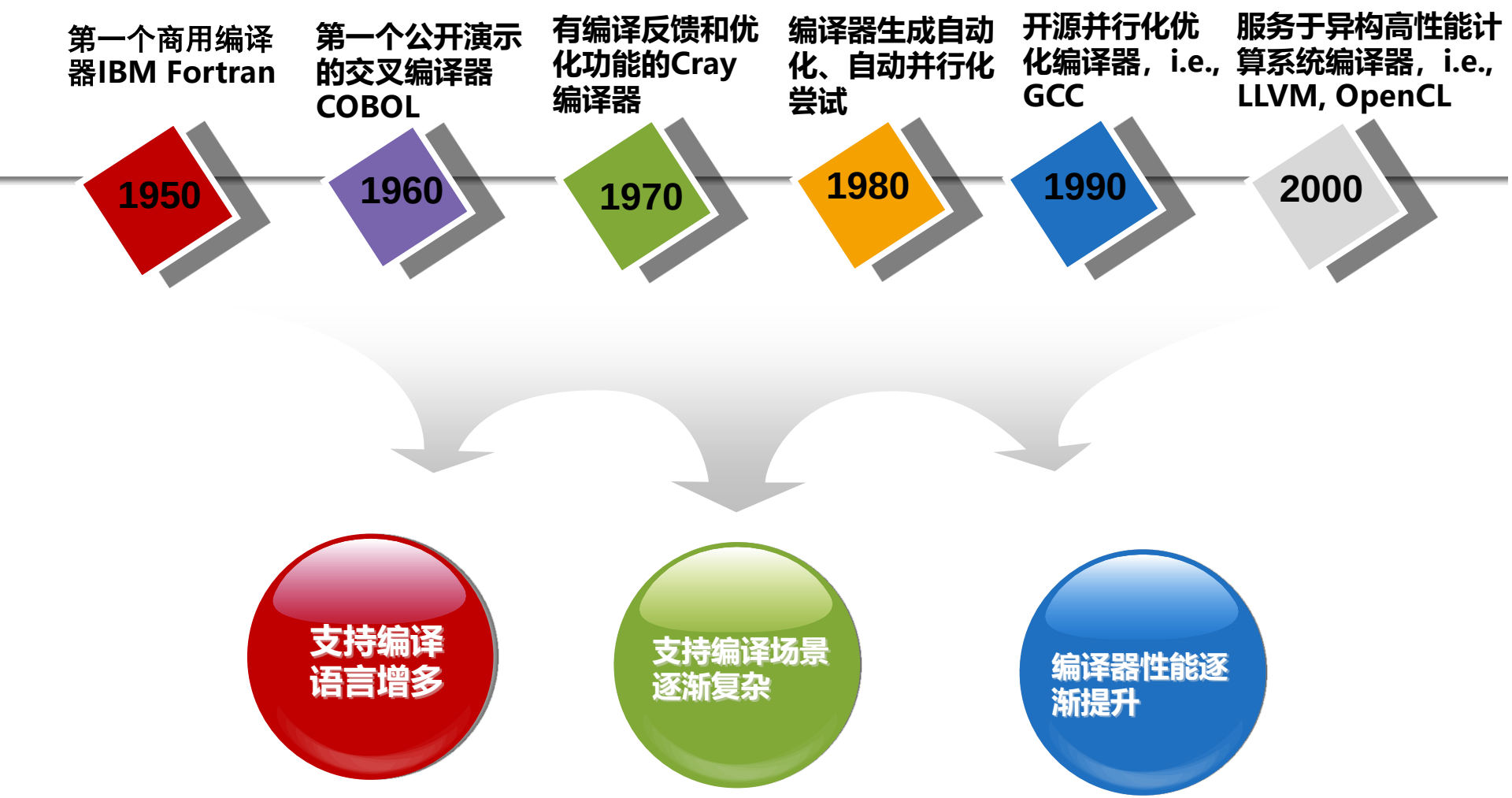
  历史上第一个实用的编译器(John Backus)：
  Fortran compiler for the IBM 704/709/7090/7094

- **FORTRAN (FORmula TRANslation)**

  第一个实用的高级语言
  擅长于数学函数运算
  常用于科学计算中

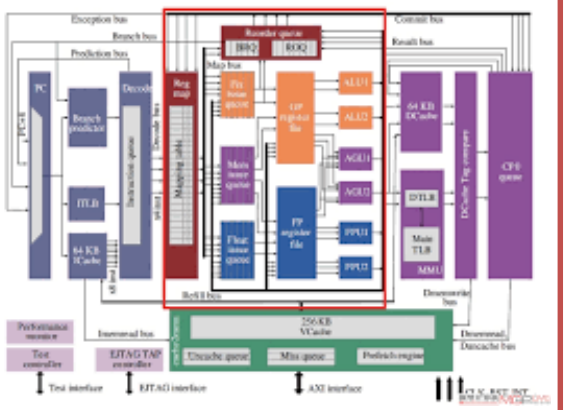
- **John Backus，引入了编译器的“阶段”或称为“遍”的概念，是编译设计的模块化的开始**

编译环境发展阶段


第一个商用编译器IBM Fortran
第一个公开演示的交叉编译器COBOL
有编译反馈和优化功能的Cray编译器
编译器生成自动化、自动并行化尝试
开源并行化优化编译器，i.e., GCC
服务于异构高性能计算系统编译器，i.e., LLVM, OpenCL
1950
1960
1970
1980
1990
2000
支持编译语言增多
支持编译场景逐渐复杂
编译器性能逐渐提升

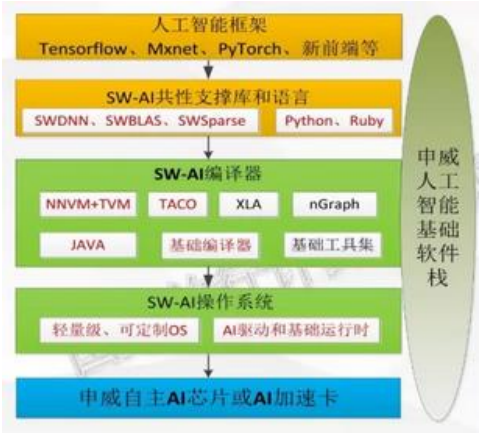
编译环境发展现状

面向嵌入式场景的编译系统



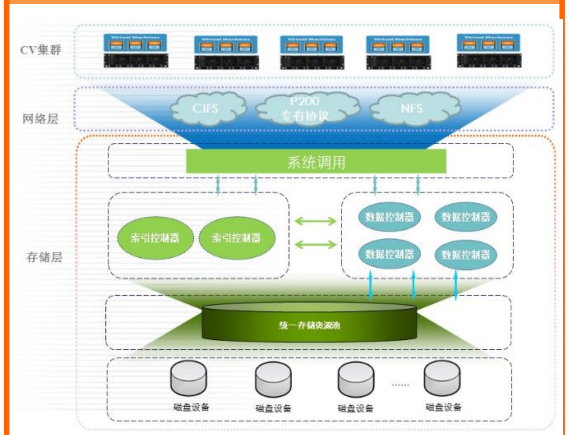
- ✓ 嵌入式ARM交叉编译工具供应商有arm公司，GNCodesourcery 公司（目前已经被Mentor收购），Linaro 公司等，支持C和C++等编程语言。
- ✓ 龙芯交叉编译器支持C，C++，Python以及Go等编程语言。

面向桌面场景的编译系统



- ✓ GCC供应商是GNU，支持C，C++，Fortran，Pascal，Java，Ada以及Go等编程语言；
- ✓ LLVM支持C，C++，Swif，Python，Ruby，Rust，Scala以及C#等语言；
- ✓ 神威编译系统支持MPI和OpenMP科学计算语言, OpenACC内核语言，Go，Java以及JS等编程语言。

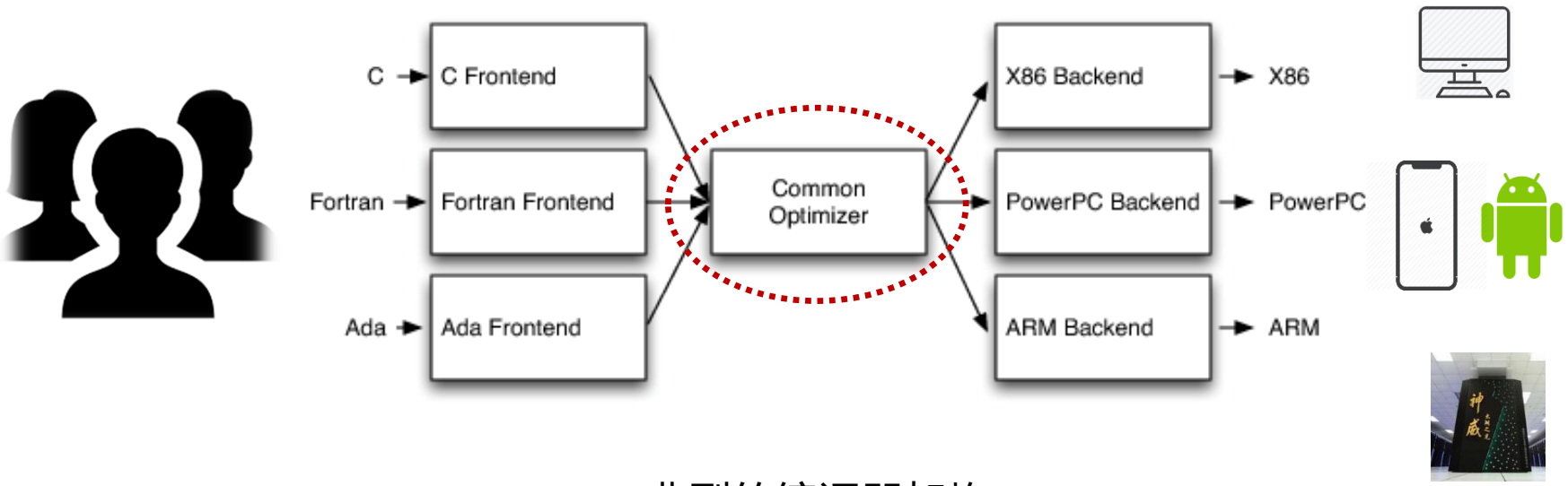
面向云环境下的编译系统



- ✓ Apache Pig编译器由Apache维护，支持Hadoop平台上Pig脚本语言的编译；
- ✓ ParallelX编译器在亚马逊 AWS GPU 云上运行，支持Java语言的编译；
- ✓ 中科曙光编译系统支持C，C++，Java以及Fortran等编程语言。

编译器与编译器故障

典型的编译器架构可分为前端、中端、后端三个部分。前端负责面向用户的高级语言的解析；中端负责对代码进行优化；后端则负责生成目标机器码。

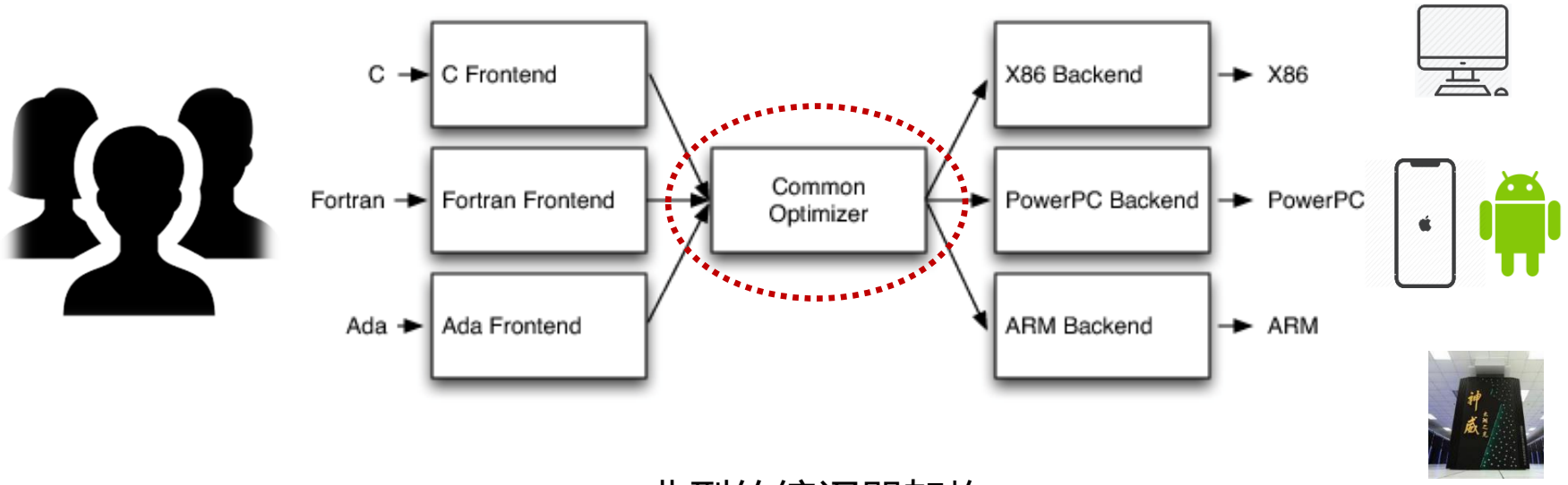


典型的编译器架构

编译器与编译器故障

- 现代编译器提供能大量的程序优化方法，
  - **~ 150 in LLVM**
  - **~ 250 in GCC**

- 以及许多标准的优化等级选项, 如 -O1, -O2, -O3, -Os
  - 每个优化等级选项表示一条由若干优化方法组成的序列

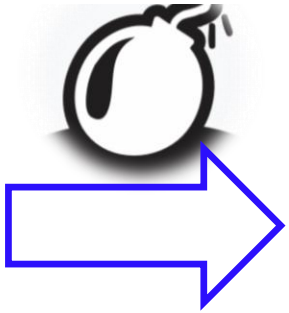


典型的编译器架构

编译器与编译器故障

编译器作为一类软件，同样不可避免地会出现故障（Bug）。

崩溃（crash）
错误编译（wrong code）
编译器性能故障（performace）
……



意料之外的程序行为
增加程序调试难度
……

编译器与编译器故障

特别是对于编译器优化，由于其种类繁多，增加了编译器出现故障的可能！

```
int a;
struct S0 {
  int f0; int f1; int f2;
};
void fn1 () {
  int b = -1;
  struct S0 f[1];
  if (a) {
    f[0] = f[b];
  }
}
int main () {
  fn1 ();
  return 0;
}
```

$ clang –O0 test.c

$ clang –O1 test.c

clang: Assertion failed.

clang: error: Aborted (core dumped)

LLVM bug 14972

编译器与编译器故障

```
int a, c, d, e = 1, f;
  int fn1 () {
  int h;
  for (; d < 1; d = e) {
    h = (f == 0) ? 0 : 1 % f;
    if (f < 1) c = 0;
    else if (h) break;
  }
}
int main () {
  fn1 ();
  return 0;
}
```

$ gcc –O0 test.c ; ./a.out

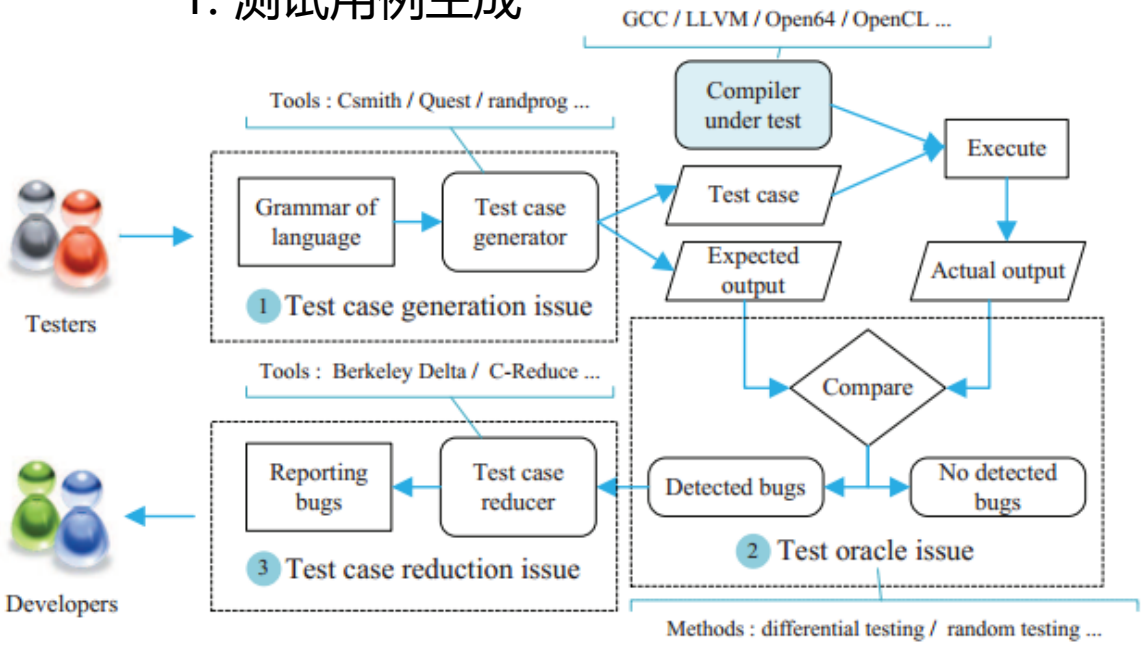
$ gcc –O2 test.c ; ./a.out

Floating point exception (core dumped)

GCC bug 61383

编译器测试

1. 测试用例生成



2. 测试Oracle问题

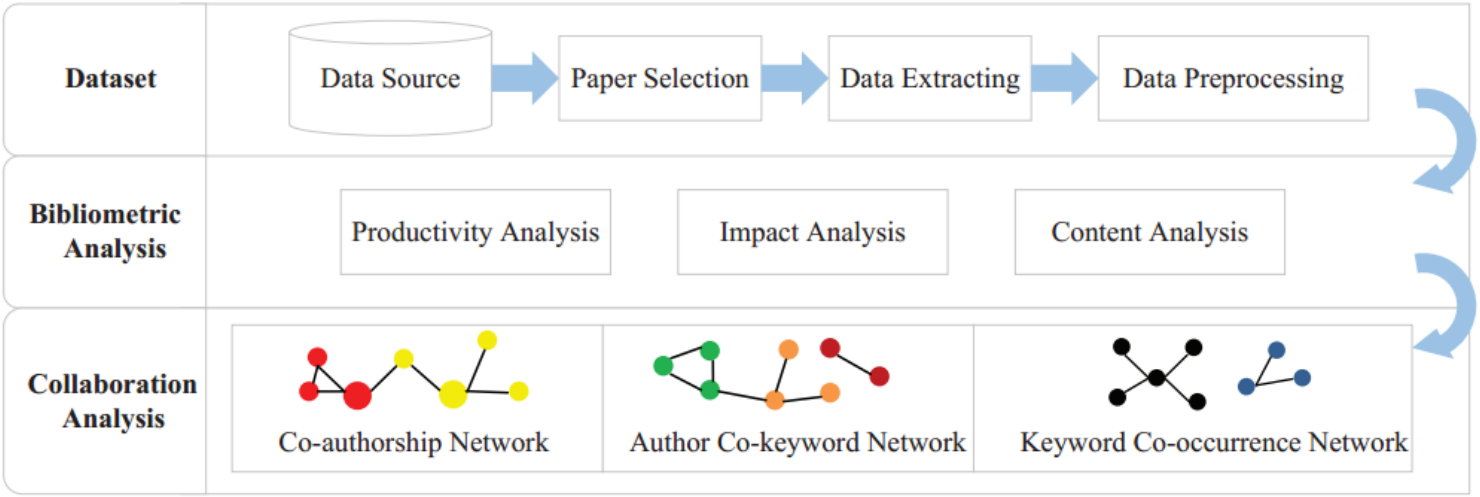
3. 测试用例约减

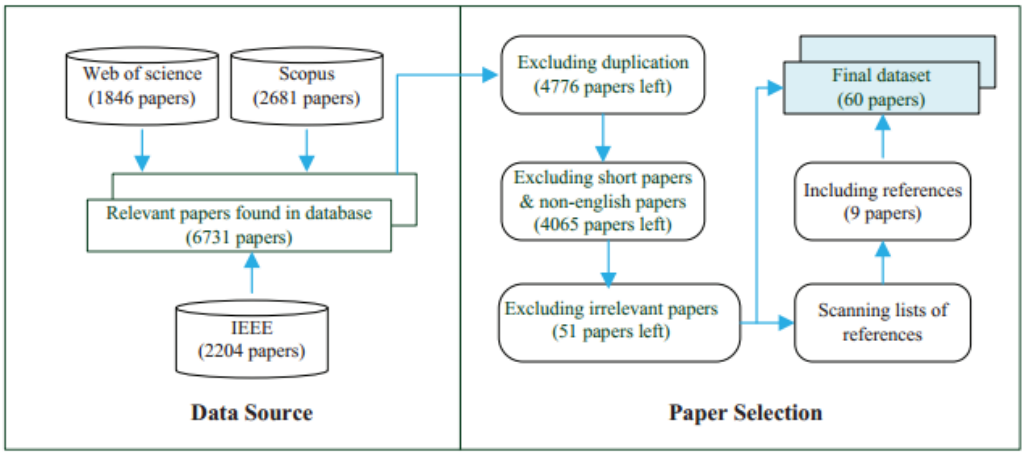
有哪些常用工具和方法呢

编译器测试领域的研究热点有哪些呢

编译器测试文献分析框架

文献分析框架



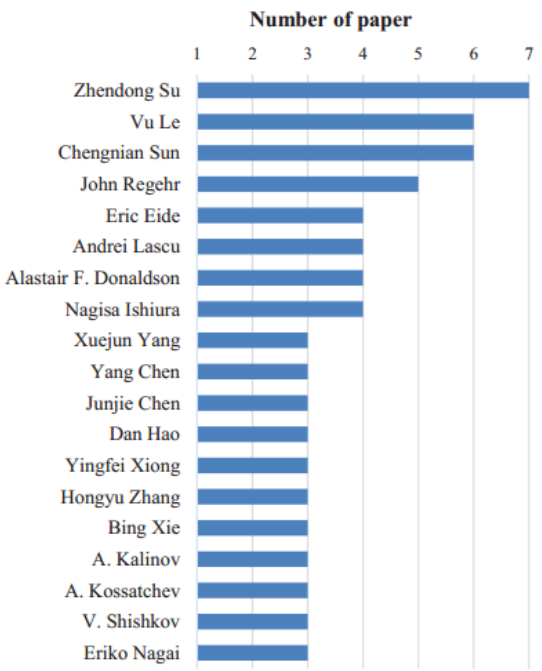


文献筛选

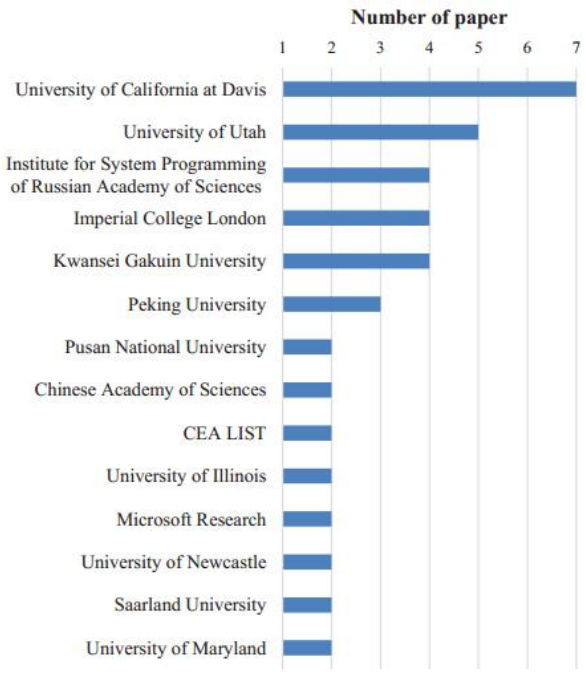
Data Item	Description
Title	Title of paper
Author	Authors'name of paper
Abstract	Abstract of paper
Keywords	Keywords presented on paper
Institution	Institution of author
Country	Country of author
Published year	Year that the paper was published
Citation	Citation number of paper
Subject	Types of compiler under test
Data generation	Tools/methods proposed to generate test case
Compiler testing technology	Types of testing method used

数据收集

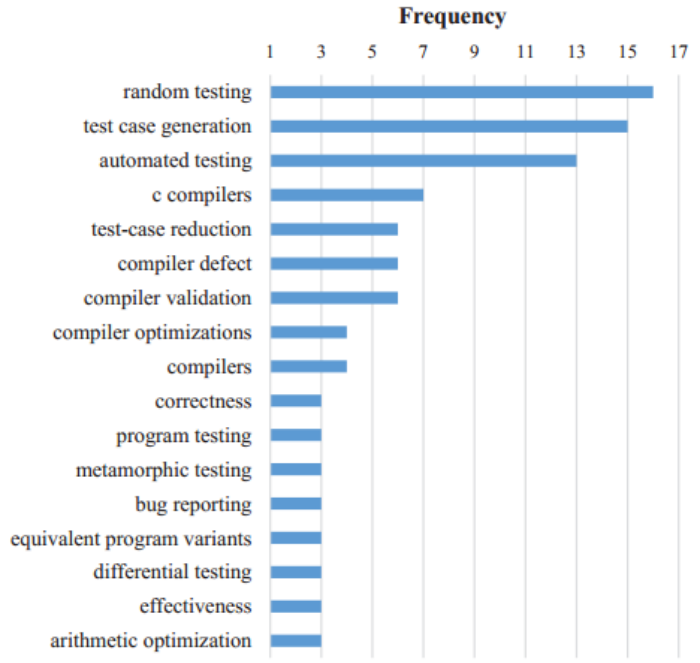
统计分析：影响力作者及机构



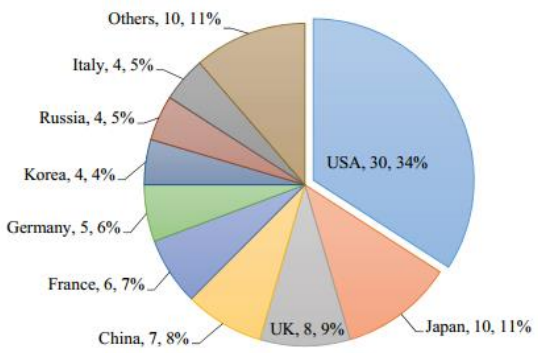
高产作者



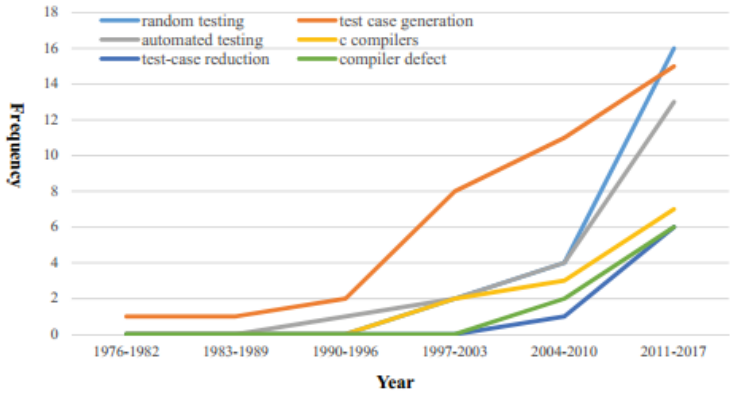
高产机构



关键词分布

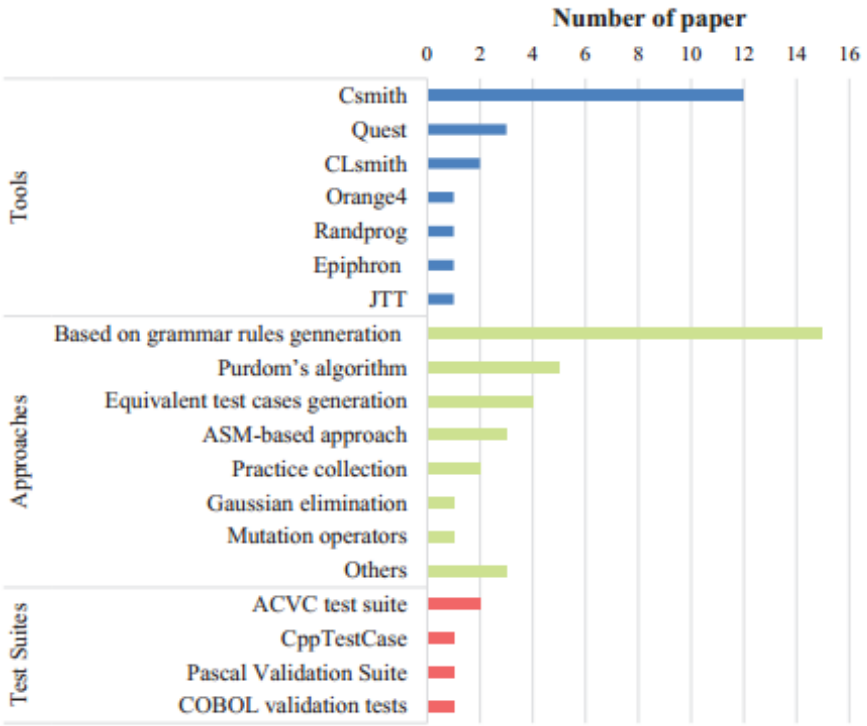


高产地区

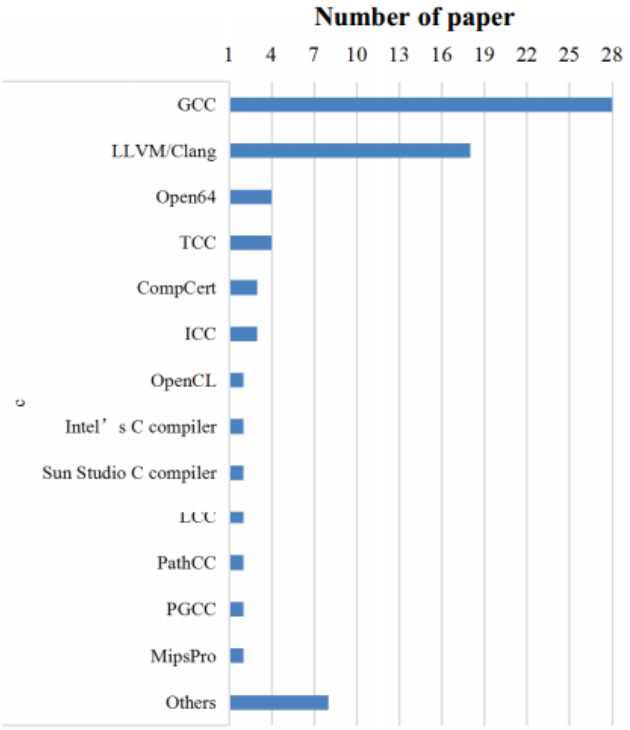


关键词趋势

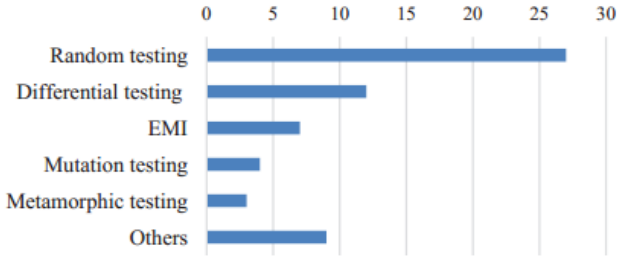
统计分析：常用工具和方法



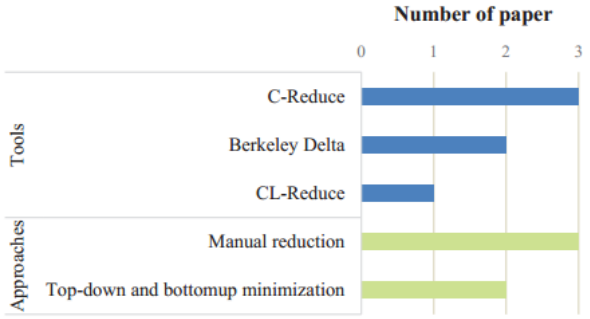
常用的测试用例生成工具



编译器分布

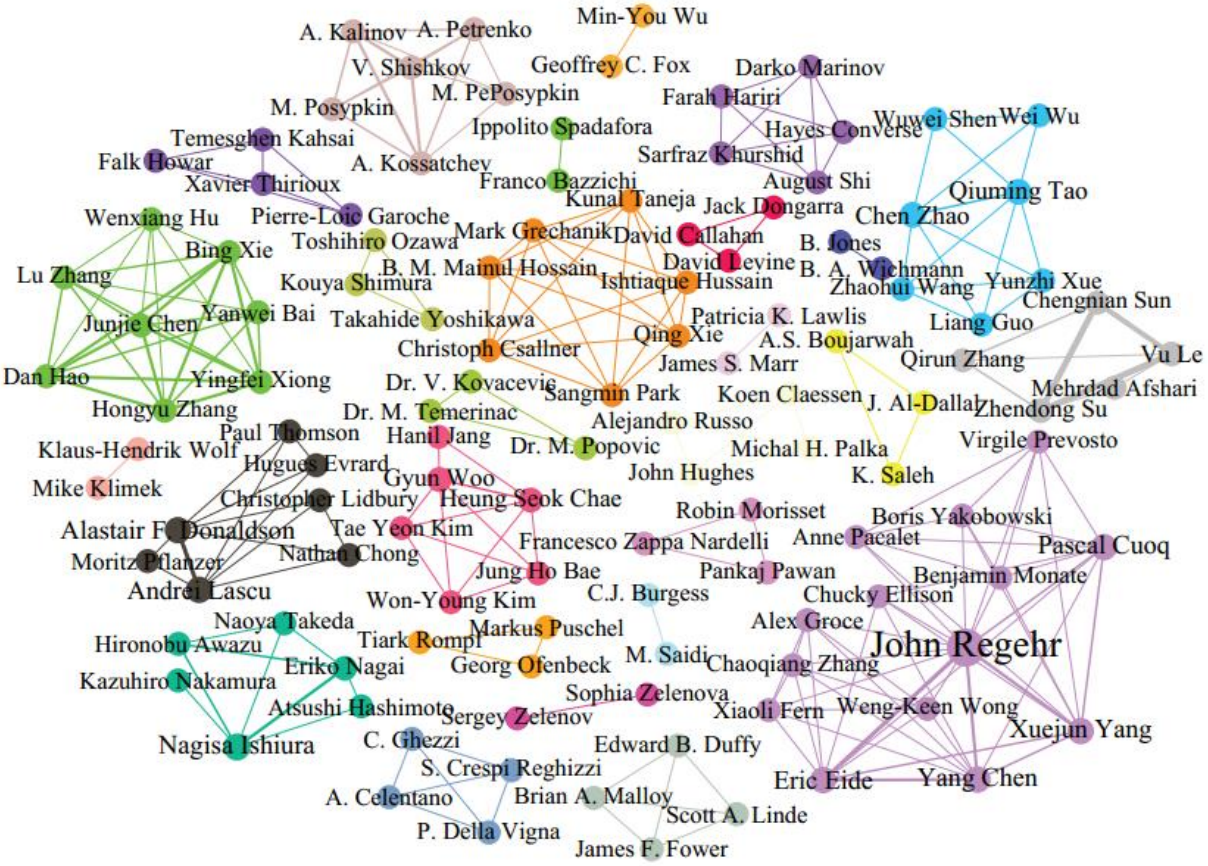


常用的编译器测试方法



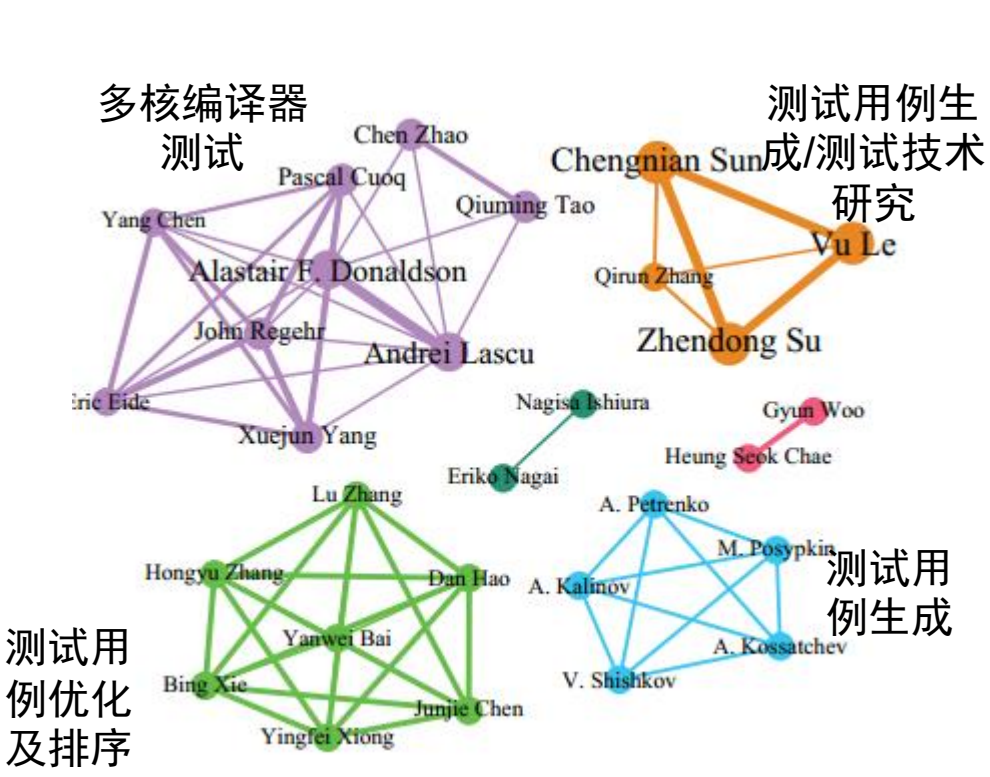
常用的测试用例约减工具

合作分析：作者合作网络

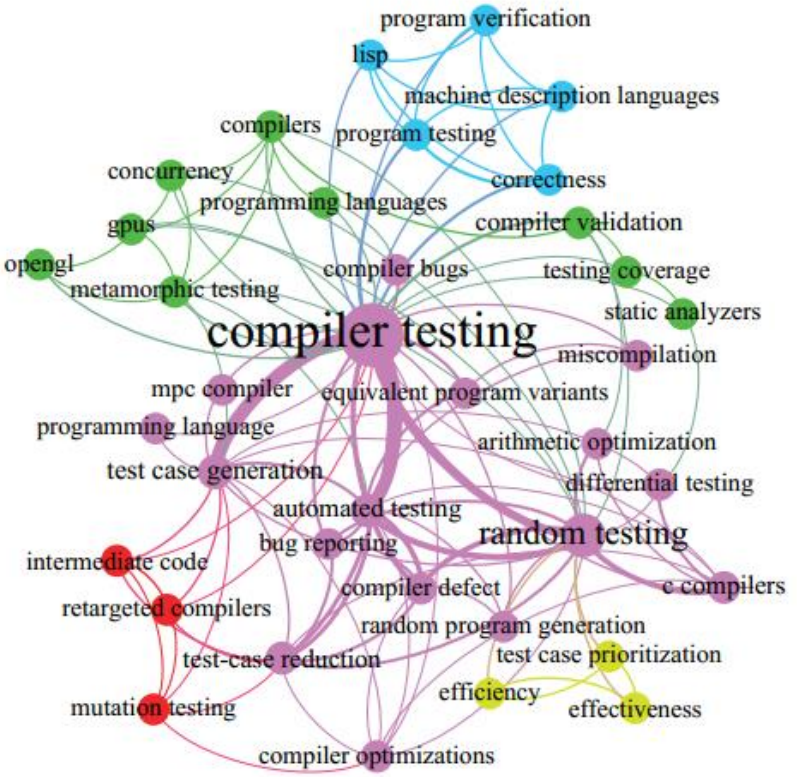


- 研究人员的合作关系固定在较小的范围内
- 研究人员趋向于同机构合作

合作分析：作者关键词网络及关键词共现网络



作者共关键词网络



关键词共现网络

- 高产作者有广泛的研究兴趣且同一机构研究热点较为相似
- 研究热点集中在编译器测试技术及测试用例生成

基于多样性引导的编译器警告测试

编译器警告介绍

- **编译器警告: 对于任何一个有潜在错误的代码片段，编译器都会输出警告信息来提醒开发者检查警告出现的代码。**
- **警告内容：通常包含警告位置，警告原因，警告选项，问题代码**

```
1  /* file=s.c */
2  int f(int a) {
3    int i = 0;
4    if (a) {
5      i++;
6    } else; /* a stray semicolon here */
7      i *= 2;
8    return i;
9  }

GCC 5.0 Output:
s.c:6:9: warning: suggest braces around empty body in an ‘else’
                                          statement [-Wempty-body]
} else; /* a stray semicolon here */
        ^
```

警告位置：6行9列

警告原因

问题代码

警告选项：[-Wempty-body]

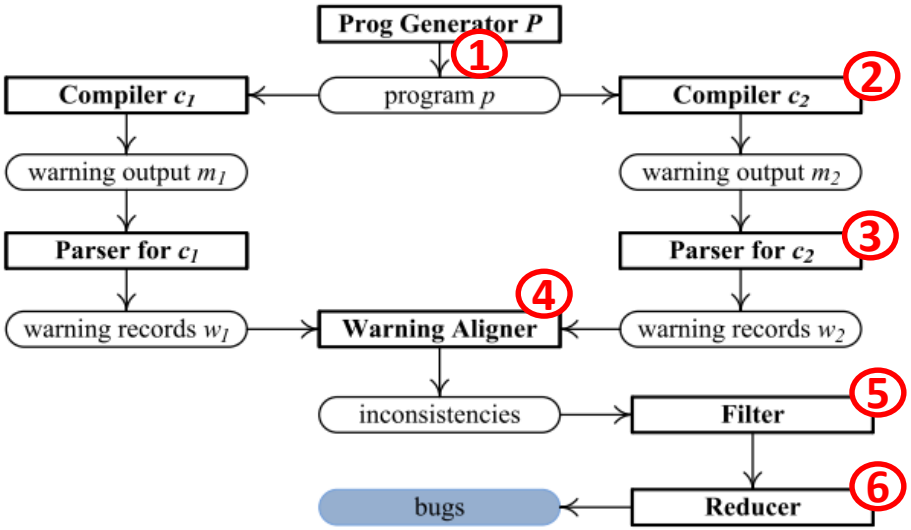
***编译器警告触发机制**：

I. **潜在的编程错误：例如代码异味**

II. **未定义行为：例如使用未初始化变量，数组越界访问**

编译器警告介绍

编译器警告测试的基本流程：



① 生成可触发编译器警告的测试用例

② 将测试用例输入到编译器中，收集警告信息

③ 提取警告信息的特定内容，并保存在数据结构中

④ 对保存的警告内容进行差异性分析

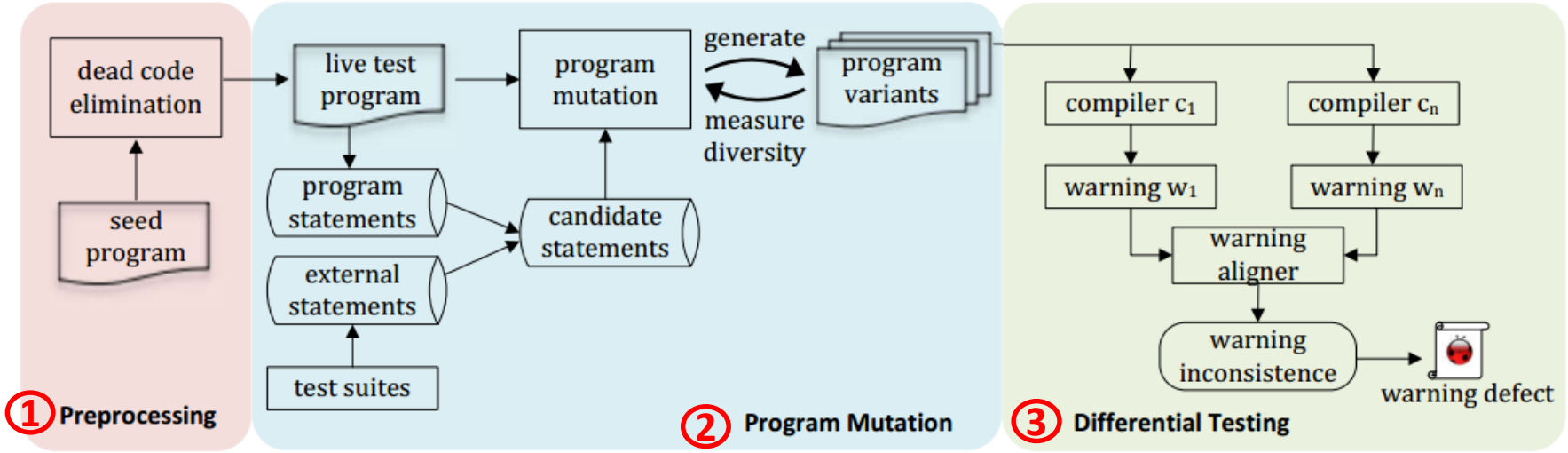
⑤ 有差异性的数据进行过滤

⑥ 对触发警告bug的测试用例进行约减，提交bug报告

测试用例生成是第一步也是关键的一步。但是目前已有的技术在构造警告敏感的程序结构仍然有欠缺

编译器警告测试框架

基于多样性引导的程序变异生成测试用例



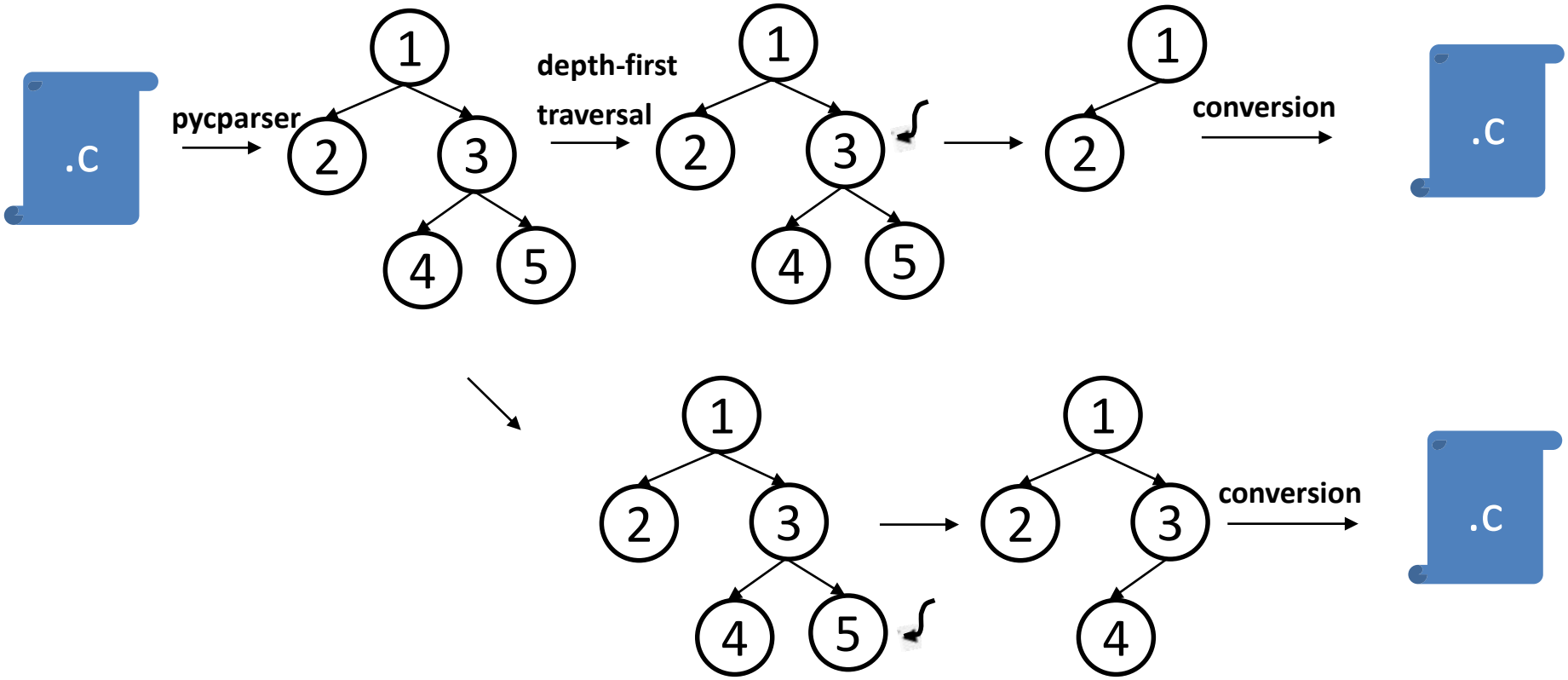
① 预处理：删除种子程序中的死代码

② 程序变异：通过添加/删除代码进行变异，在变异过程中进行多样性引导

③ 差分测试：将测试用例输入到不同编译器中进行警告测试

编译器警告测试框架

程序变异：删除操作



编译器警告测试框架

程序变异：删除因子举例

TABLE 1
Typical pruning mutators

Mutation type	Typical mutators	Example (Code before mutation →Code after mutation)	Warning diagnostic
Prune variable	Removing the attributes of a variable, e.g., qualifiers, modifiers, and types;	static float g_1; → static g_1;	warning: type specifier missing, defaults to "int" [-Wimplicit-int]
Prune operator	Removing a operator and its attributes, e.g, unary operators, binary operators, and ternary operators;	int g_1= 1>0? 1 :0; → int g_1= 1>0? :0;	warning: ISO C forbids omitting the middle term of a ?: expression [-Wpedantic]
Prune expression statement	Removing the expression and its attributes, e.g., assignments and arithmetic expressions;	int g_1 = 1 ; → ;	warning: empty expression statement has no effect [-Wextra-semi-stmt]
Prune Control structure	Removing the structure and its attributes, e.g., the loop structure, the branch structure, and the jump structure;	if (g_1) goto Label ; → if (g_1);	warning: suggest braces around empty body in an "if" statement [-Wempty-body]
Prune function	Removing the function declaration and the whole function body;	int func_1(void) ; → ;	warning: no previous prototype for function "func_1" [-Wmissing-prototypes]

编译器警告测试框架

程序变异：插入操作

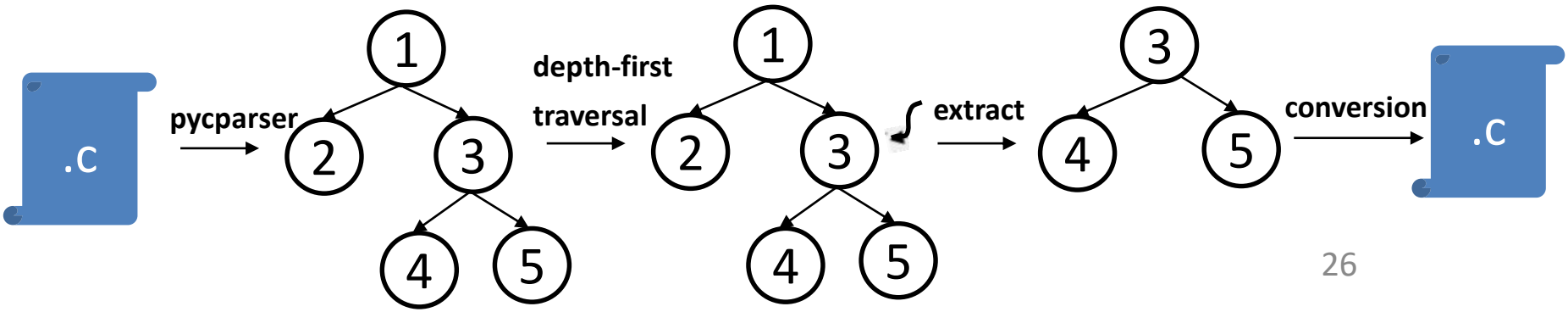
① 代码块切片

② 重复代码块删除

③ 插入过程重命名

① 代码块切片

- **数据源**：从2种测试用例中提取代码块
  1）Epiphron生成的测试用例　　2）GCC/Clang开源数据集
- **提取代码结构：**表达式语句/循环语句/跳转语句/分支语句/函数块
- **结果**：最终得到超过30,000个代码语句



编译器警告测试框架

程序变异：插入操作

② 重复代码块删除

- **流程：**提取每个片段的骨架，如图所示，(a)和(c)有相同的骨架，我们认为这两个程序是等价的，数据库中只保留该骨架的一个程序即可。提取每个片段的骨架，如图所示，(a)和(c)有相同的骨架，我们认为这两个程序是等价的，数据库中只保留该骨架的一个程序即可。

```
if (a > b)
   a = a + fun_1(b)
else
   a = fun_2(b)
```
(a) statement S1

```
if (□ > □)
   □ = □ + □(□)
else
   □ = □(□)
```
(b) Skeleton S1

```
if (c > d)
   c = c + fun_2(d)
else
   c = fun_1(d)
```
(c) statement S2

```
if (□ > □)
   □ = □ + □(□)
else
   □ = □(□)
```
(d) Skeleton S2

③ 插入过程重命名

- **目的**：为了保证插入的程序是有效的，我们需要对插入的代码片段进行重命名，使得其符合插入点的上下文环境。
- **流程：**我们提起插入点上下文的全局变量和局部变量。对于需要重命名的变量，我们优先将其重命名为数据类型一致的局部变量名，如若没有合适的，再选择数据类型一致的全局变量名进行重命名。如果无法重命名，我们丢掉这个代码片段，重新选择一个代码片段进行插入。

编译器警告测试框架

程序变异：插入因子举例

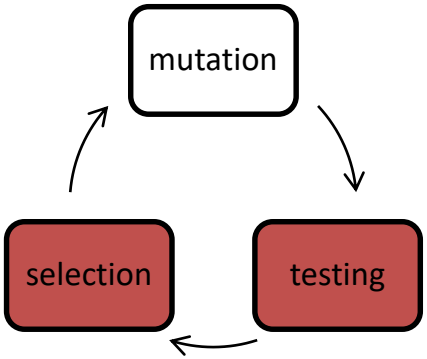
TABLE 2
Typical inserting mutators

Mutation type	Typical mutators	Example (Code before mutation →Code after mutation)	Warning diagnostic
Insert expression statement	Inserting a single statement, e.g., an assignment and an arithmetic expression;	int main(){int l_1; ...} → int main(){int l_1=1; int l_2=(0!=((-1)|l_1)); ...}	warning: bitwise comparison always evaluates to true [-Wtautological-compare]
Insert control structure	Inserting a type of control structure, e.g., the loop structure and the jump structure;	int func_1(){int l_1; ...} →int func_1(){int l_1; ... return l_1; }	warning: variable "l_1" is uninitialized when used here [-Wuninitialized]
Insert function	Inserting the function definition and the corresponding function call statement;	int main(){... unsigned int l_1; ...} → int func_1(int p_1){... return p_1;} int main(){... unsigned int l_1; l_1=func_1(0); ...}	warning: implicit conversion changes signedness: "int" to "unsigned int" [-Wsign-conversion]

编译器警告测试框架

程序变异：基于多样性引导的变异因子选择

- **不同的变异因子构造警告敏感的能力不同**
- **频繁生成有效且多样化程序的变异因子应该有较大的概率被选择**



An MCMC Sampling Method

① 每个变异因子被选择之后计算它的分数，并根据分数排序，生成变异因子的排序列表

$$Score(Mut) = (\frac{1}{n}\sum_{i=1}^{n} Dist(P, P_i)) * succ(Mut)$$

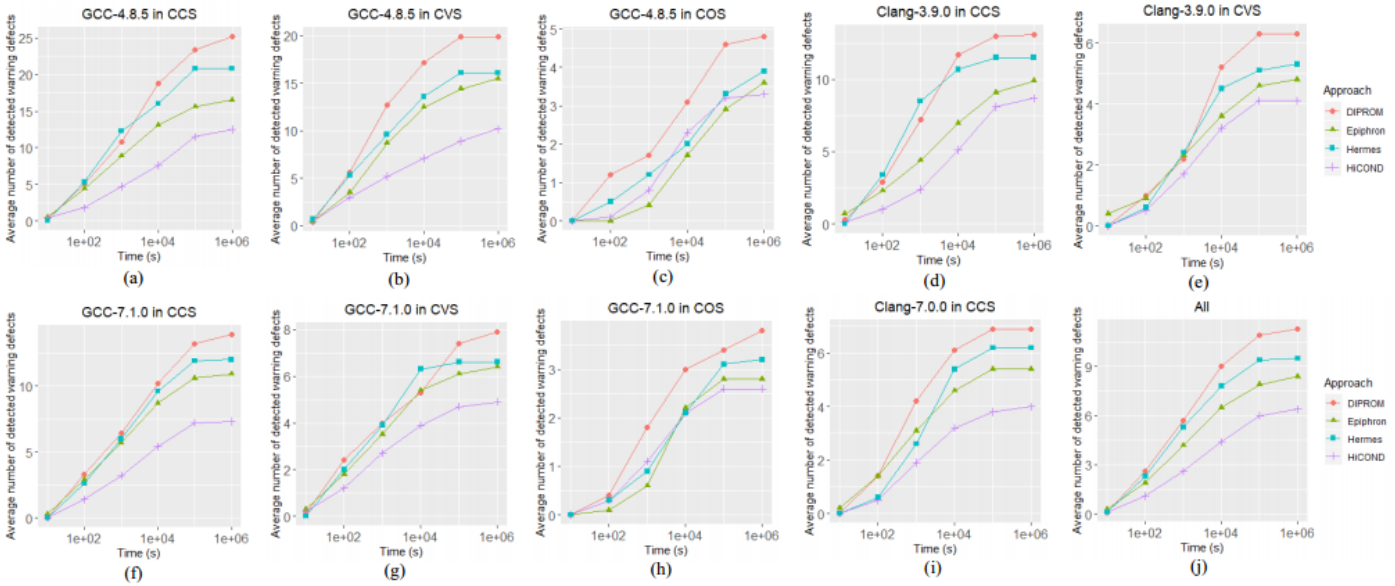
② 假设前一次变异因子是Mut_a，当前选择的变异因子是Mut_b，判断Mut_b是否用于当前变异的依据是：

$$Pb(M_b|M_a) = min(1, \frac{Pr(Mut_b)}{Pr(Mut_a)}) = min(1, (1-p)^{k_b - k_a})$$

如果Mut_b在排序列表中的位置比Mut_a靠前(即$K_b<K_a$)，接受当前变异因子；否则，Mut_b仍有 $(1-p)^{Kb-Ka}$的概率被接受用于当前的变异。

分析结果I：我们的方法 vs 对比算法

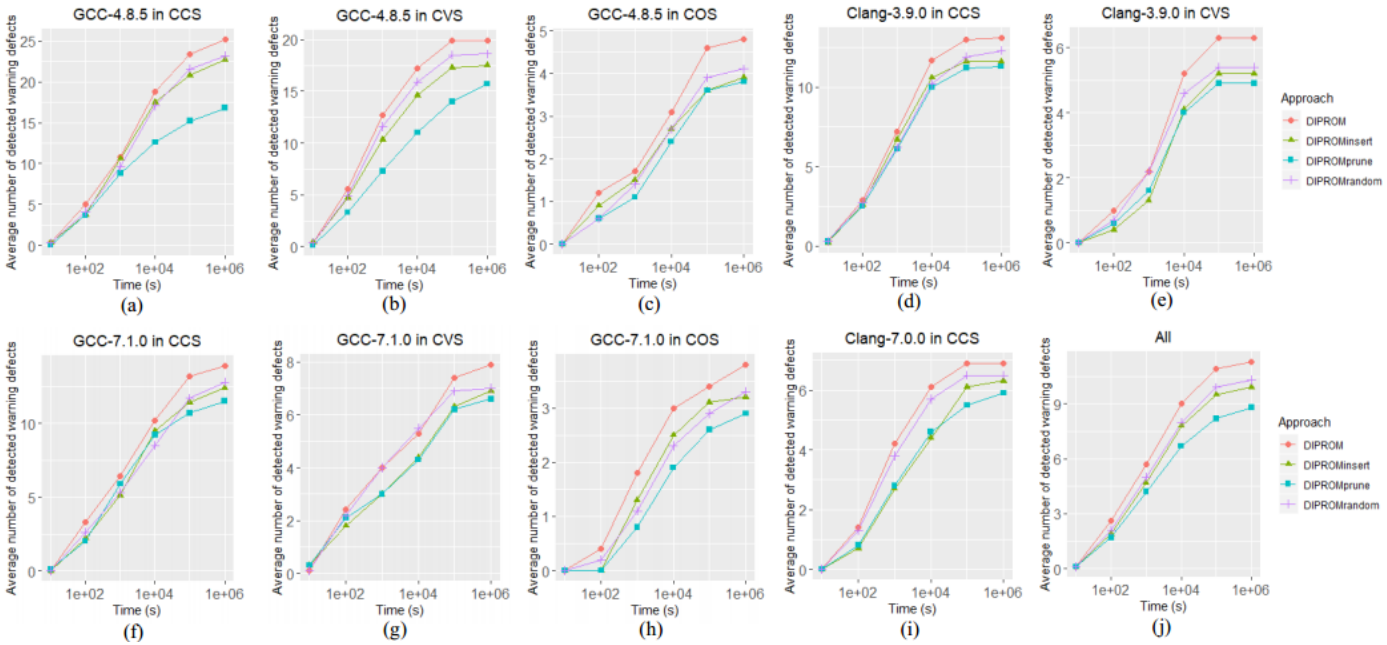
Subject	Scenario	DIPROM		HiCOND					Epiphron					Hermes				
		#Ave. bugs	#TP. (*10^3)	#Ave. bugs	imp.	#TP. (*10^3)	*P*-value	Effect size	#Ave. bugs	imp.	#TP. (*10^3)	*P*-value	Effect size	#Ave. bugs	imp.	#TP. (*10^3)	*P*-value	Effect size
GCC-4.8.5	CCS	25.2	53.39	12.5	101.60%	63.49	<0.001	1.000	16.5	52.73%	63.75	<0.001	1.000	20.8	21.15%	54.29	<0.001	1.000
	CVS	19.9	52.67	10.2	95.10%	53.55	<0.001	1.000	15.5	28.39%	60.98	<0.001	1.000	16.1	23.60%	53.45	<0.001	1.000
	COS	4.8	17.11	3.3	45.45%	17.48	<0.001	0.960	3.6	33.33%	19.22	<0.001	0.920	3.9	23.08%	21.74	0.004	0.830
GCC-7.1.0	CCS	13.9	60.54	7.3	90.41%	68.64	<0.001	1.000	10.9	27.52%	70.78	<0.001	1.000	12.0	15.83%	61.95	0.004	0.855
	CVS	7.9	57.69	4.9	61.22%	65.85	<0.001	1.000	6.4	23.44%	68.04	<0.001	0.925	6.6	19.70%	60.21	0.002	0.870
	COS	3.8	23.72	2.6	46.15%	25.84	0.002	0.860	2.8	35.71%	26.61	0.003	0.845	3.2	18.75%	23.31	0.016	0.750
Clang-3.9.0	CCS	13.1	51.15	8.7	50.57%	59.42	<0.001	1.000	9.9	32.32%	60.21	<0.001	1.000	11.5	13.91%	52.66	<0.001	0.950
	CVS	6.3	51.60	4.1	53.66%	58.61	<0.001	0.980	4.8	31.25%	60.32	<0.001	0.935	5.3	18.87%	51.57	0.002	0.860
Clang-7.0.0	CCS	6.9	54.89	4.0	72.50%	62.65	<0.001	1.000	5.4	27.78%	63.50	<0.001	0.975	6.2	11.29%	55.28	0.001	0.850
	Total	101.8	422.76	57.6	76.74%	475.53	-	-	75.8	34.30%	493.41	-	-	85.6	18.93%	434.46	-	-



我们的方法(DIPROM)优于对比算法：检测出更多的bug，且用时较短。

分析结果II：我们的方法 vs 算法变体

Subject	Scenario	DIPROM		$DIPROM_{prune}$					$DIPROM_{insert}$					$DIPROM_{random}$				
		#Ave. bugs	#TP. ($*10^3$)	#Ave. bugs	imp.	#TP. ($*10^3$)	*P*-value	Effect size	#Ave. bugs	imp.	#TP. ($*10^3$)	*P*-value	Effect size	#Ave. bugs	imp.	#TP. ($*10^3$)	*P*-value	Effect size
GCC-4.8.5	CCS	25.2	53.39	16.8	50.00%	56.13	<0.001	1.000	22.7	11.01%	50.23	<0.001	1.000	23.3	8.15%	52.64	0.002	0.870
	CVS	19.9	52.67	15.7	26.75%	54.18	<0.001	1.000	17.3	15.03%	49.89	<0.001	0.940	18.5	7.57%	52.18	0.001	0.870
	COS	4.8	17.11	3.8	26.32%	17.26	<0.001	0.920	3.9	23.08%	16.71	0.001	0.870	4.1	17.07%	16.70	0.012	0.770
GCC-7.1.0	CCS	13.9	60.54	11.5	20.87%	64.47	0.001	0.905	12.4	12.10%	57.26	0.006	0.803	12.8	8.59%	59.83	0.019	0.770
	CVS	7.9	57.69	6.6	19.70%	57.88	0.001	0.910	6.9	14.49%	55.67	0.007	0.815	7.0	12.85%	58.04	0.015	0.775
	COS	3.8	23.72	2.9	31.03%	25.63	0.001	0.865	3.2	18.75%	22.53	0.016	0.750	3.3	15.15%	23.25	0.045	0.700
Clang-3.9.0	CCS	13.1	51.15	11.3	15.93%	53.22	<0.001	0.940	11.6	12.93%	49.60	<0.001	0.940	12.2	7.38%	51.06	0.013	0.780
	CVS	6.3	51.60	4.9	28.57%	52.09	<0.001	0.930	5.2	21.15%	48.76	0.001	0.890	5.4	16.67%	49.83	0.018	0.765
Clang-7.0.0	CCS	6.9	54.89	5.9	16.95%	58.12	<0.001	0.970	6.3	9.52%	51.70	0.012	0.755	6.4	7.81%	54.07	0.013	0.750
	Total	101.8	422.76	79.4	28.21%	438.98	-	-	89.5	13.74%	402.35	-	-	93.0	9.46%	417.60	-	-



我们的方法(DIPROM)比算法变体检测出更多的bug

分析结果III：方法应用

	GCC	Clang	Total
Reported	6	2	8
Confirmed	4	1	5
Pending	0	1	1
Duplicate	1	0	1
Suspended	1	0	1

开发版本的GCC和Clang测试结果

我们的方法(DIPROM)检测出8个警告bug，其中5个被开发者确认

数组引用越界导致的GCC警告bug

```
1 // file = s.c
2 #include <stdio.h>
3 int main(){
4   int a[1]={0};
5   printf("%d", a[3]);//no warning here
6 }
```

```
Clang 10 outputs:
s.c:5:16: warning: array index 3 is past the end
    of the array (which contains 1 element) [-
    Warray-bounds]
  printf("%d", a[3]);
```

Fig. 9. GCC missing warning defect #92378 (*https://gcc.gnu.org/bugzilla/show_bug.cgi?id = 92378*).

重言式比较导致的Clang警告bug

```
1 // file = s.c
2 int main(){
3   int a;
4   int b=(0!=((-1)|((a=1)==1)));//no warning here
5   return 0;
6 }
```

```
GCC 10 outputs:
s.c:4:12: warning?bitwise comparison always
    evaluates to true [-Wtautological-compare]
4 | int b=(0!=((-1)|((c = 1) == 1)));
```

Fig. 10. Clang missing warning defect #92479 (*https://bugs.llvm.org/show_bug.cgi?id = 92479*).

总结

为了测试编译器警告机制中存在的问题，我们

- 提出了一种基于多样性引导的变异算法，用于生成大量警告敏感的程序结构来帮助测试编译器警告
- 我们在两个常用编译器GCC和Clang进行验证，实验结果显示我们的方法由于对比算法18.93%~76.74%
- 我们在开发版本上验证我们的方法，找到了8个新版本的编译器bug。

谢 谢!

大连理工大学　江 贺