C언어를 반석 위에

허충길
소프트웨어 원리 연구실
서울대학교

@Rosaec Workshop

C언어 퀴즈 1

아래 함수에서 잘못된 점은?

```
/* arr의 크기는 항상 size 이상임을 가정한다. */
int foo(int *arr, int size) {
   int sum = 0;
   int *end;

   end = arr + size;
   for (; arr < end; arr += 2) {
      sum += *arr;
   }
   return sum;
}
```

C언어 퀴즈 2

아래 프로그램에서 나올 수 있는 가능한 출력은?

```
int main() {
    int x = foo();
    printf(x <  0 ? "a\n" : "b\n");
    printf(x >= 0 ? "a\n" : "b\n");
    return 0;
}
```

C언어 퀴즈 3

아래 프로그램의 가능한 수행 결과는?

```
void foo() {
  int a = 1;
  while (a) { }
}

void bar() {
  int i;
  foo();
  for (i = 0; i < 1; ++i) { }
}

int main() {
  bar();
  printf("boom!\n");
}
```

C언어 퀴즈 4

아래 프로그램은 정상일까 아닐까?

```
void foo(char* buf, int len) {
  /* Wrapping checks */
  if (buf + len < buf) {
    printf("buf is too big.\n");
    return;
  }

  /*
     do something using buf
  */
}
```

C언어 퀴즈 5

아래 프로그램의 수행 결과는?

```c
union int_float {
   int a;
   float b;
};

float foo(int* a, float* b) {
   *b = 10.0f;
   *a = 0;
   return *b + *b;
}

int main() {
   union int_float u;
   printf("%f\n", foo(&u.a,&u.b));
}
```

현재 C 언어는 기반이 약하다

- C언어 정의의 문제점
  - ISO C standard는 700 페이지
  - 너무 복잡
  - 너무 모호

- C컴파일러 (개발자)의 문제점
  - 컴파일러: 많은 버그들을 포함

  - 컴파일러 개발자:
    컴파일러 최적화를 합리화하기 위해
    C standard committee로 활동하여
    C언어의 정의를 너무 지저분하게 만들어
    C 프로그램 개발자들을 괴롭힘.

우리의 연구: C언어를 반석 위에!

- 간단하고 엄밀한 C언어 의미를 정의
  - C언어의 Operational Semantics를 Coq에서 정의
  - 간단하고
  - 엄밀하게
- C 컴파일러의 버그를 박멸
  - Coq으로 검증된 검산기(Validator)를 개발
  - LLVM 컴파일러(100만줄)를 사용할 때 사용자가 오류가 없음을 확신할 수 있게.

간단하고 엄밀한 C언어 의미 정의

- 핵심은
  - 간단하면서도
  - 컴파일러의 거의 모든 최적화를 다 허용

- 이를 위해서는
  - 수학적으로 훌륭한 모델을 개발
  - 컴파일러를 잘 수정

현재 ISO C standard의 이슈들

- 여러 가지 이슈들
  - Pointer-Integer Casting
    - We solved this! (hopefully appear at PLDI 2015)
  - Dangling-pointer manipulation
  - Undefined Value
  - Silent non-termination
  - Integer Overflow
  - Strict Aliasing
    - 너무 지저분한 기능, 많은 경우 사용하지 않음
    - 우리는 당분간 지원하지 않을 계획임
  - ⋮

Pointer-Integer Casting의 문제

아래 컴파일러 최적화가 적합할까?

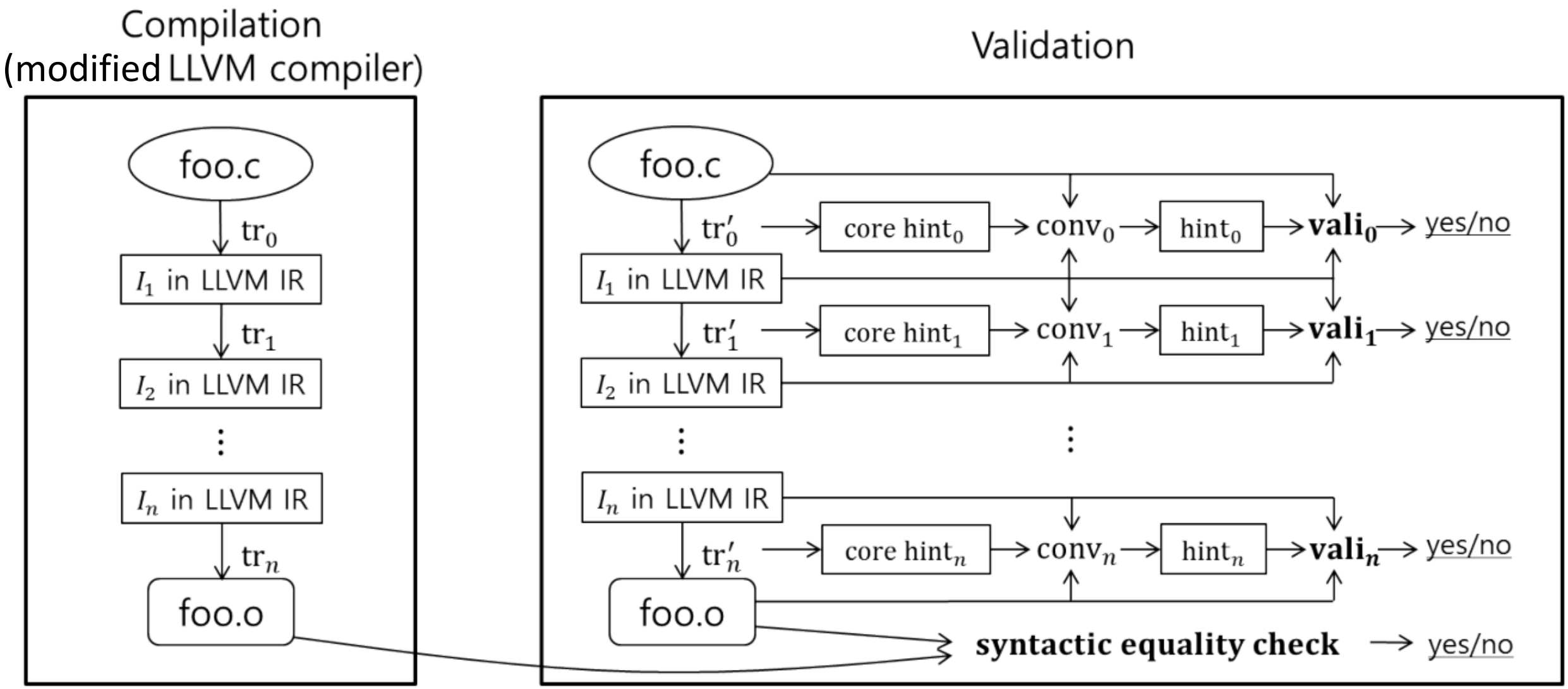
```
extern void g(void);
int f(void) {
  int a = 0;
  g();
  return a;
}
```

→

```
extern void g(void);
int f(void) {
  int a = 0;
  g();
  return 0;
}
```

자세한 내용은 강지훈 학생의 톡(1)에서!!

LLVM 컴파일러를 위한 검증된 검산기



Scalability를 위한 우리의 접근 방법

- 일반적이고 강력한 검산기
  - Relational Hoare logic에 기반한 일반적인 검산기
  - 이 검산기 하나면 끝!
    - 프로그램의 CFG (Control Flow Graph)를 바꾸지 않는 모든 최적화를 검산할 예정
    - LLVM 컴파일러에는 대략 120개의 최적화가 있음.
    - 그 중 100개 이상이 CFG를 바꾸지 않음.

- 검산 시간이 문제
  - 검산 시간은 컴파일 시간의 100배 이상 걸릴지도
  - 하지만, 컴파일러는 1000번 이상의 최적화를 수행
  - 이들 최적화들은 모두 독립적으로 검산 가능
  - 따라서, Core 100개면 100배의 Speed Up!

결론

- 간단하고 엄밀한 C언어 의미 정의
  - 엄밀
    - Coq으로 정의된 Operational Semantics
  - 간단
    - 직관에 부합하는 아주 간단한 정의

- 성능 좋고 믿을 수 있는 컴파일러
  - LLVM 컴파일러 수정해서 올바른 최적화만 남김
  - Original Compiler와 거의 비슷한 성능
  - 검증된 검산기를 이용해서 버그 박멸

광고: CompCert 컴파일러 증명 개선 및 버그 최초 발견!
자세한 내용은 강지훈 학생 톡(2)에서!!