올바른 컴파일러 최적화 검산을 위한 지향성 퍼징

권재성, 장봉준, 허기홍

Programming Systems Laboratory

KAIST

배경

올바른 최적화는 프로그램의 **의미**를 보존해야 한다

- 최적화 버그는 프로그램의 의미를 원래와 달라지게 할 수 있다

```
int foo () {
  signed char x = 1;
  unsigned char y = 255;
  return x > y;
}
```

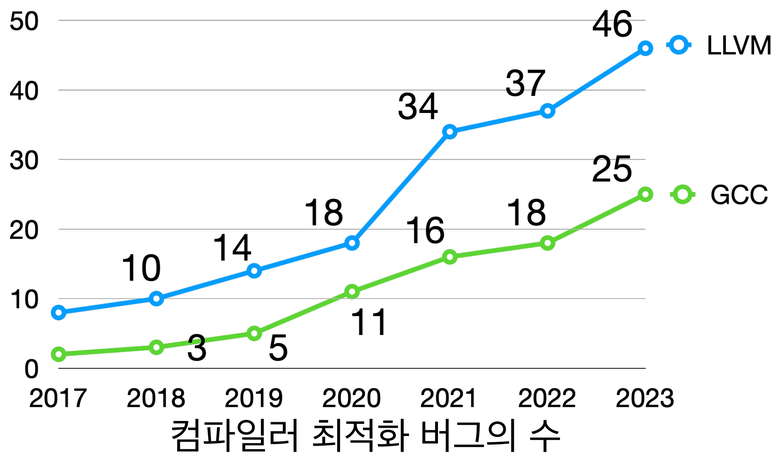
→ 최적화 →

```
int foo () {
  return 1;
}
```

최적화 버그 예시

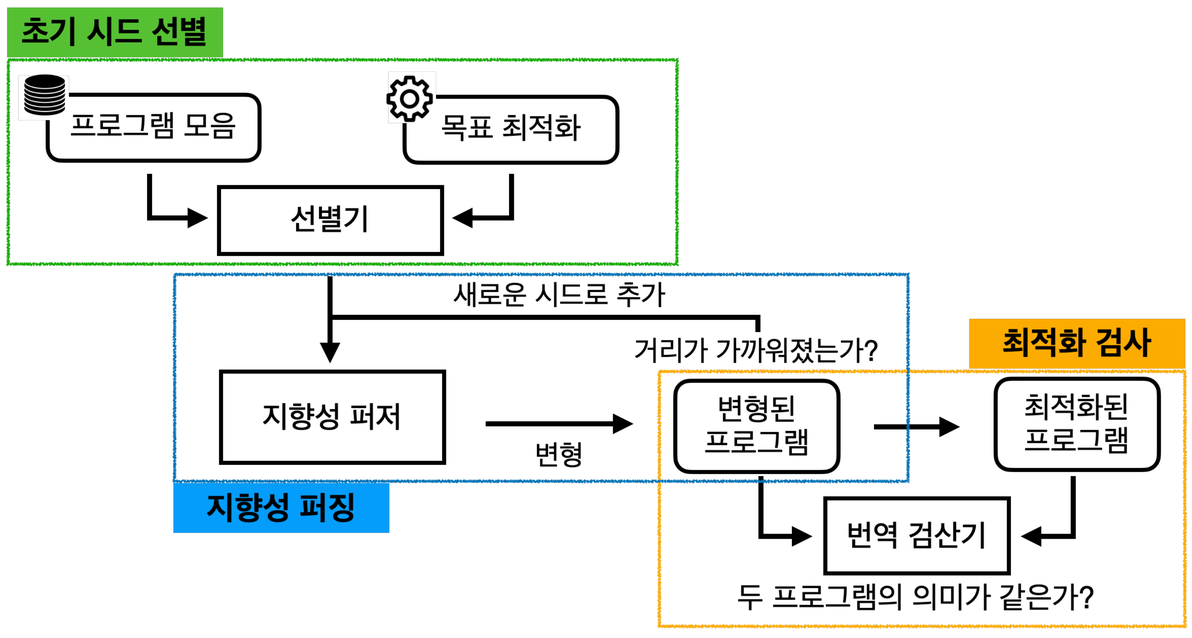
목표: 원하는 최적화를 집중적으로 검사하는 지향성 퍼저

- 컴파일러의 최적화는 크고 복잡하기에 정적 분석과 검증이 어려움
- 특정 최적화를 일으키는 **다양한** 프로그램을 효과적으로 생성
- 특정 최적화를 검사하기 위한 **지향성** 퍼징



컴파일러 최적화 버그의 수

전체 모식도



지향성을 위한 거리 점수

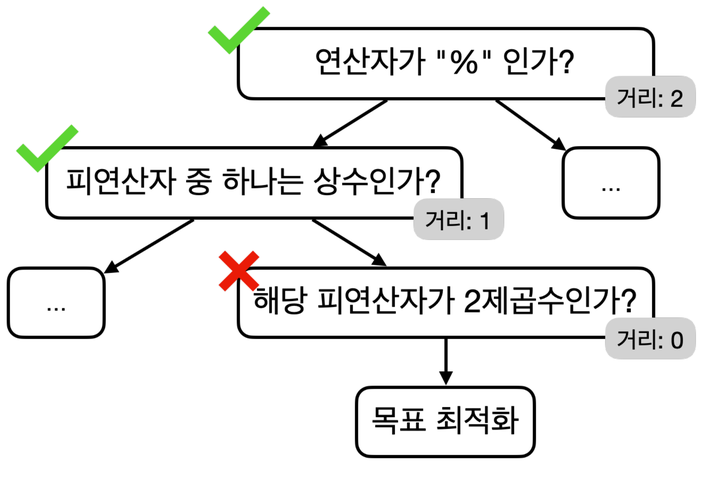
지향성 점수: 프로그램이 최적화 조건을 얼마나 만족하는가?

- 컴파일러는 **중첩된 조건문**으로 최적화 조건을 검사
- **거리 점수**: 목표까지 조건문을 얼마나 더 통과해야 하는가?

목표 최적화: $X \,\%\, 2^C \rightarrow X \,\&\, (2^C - 1)$

```
int foo (int X) {
  return X % 5;
}
```

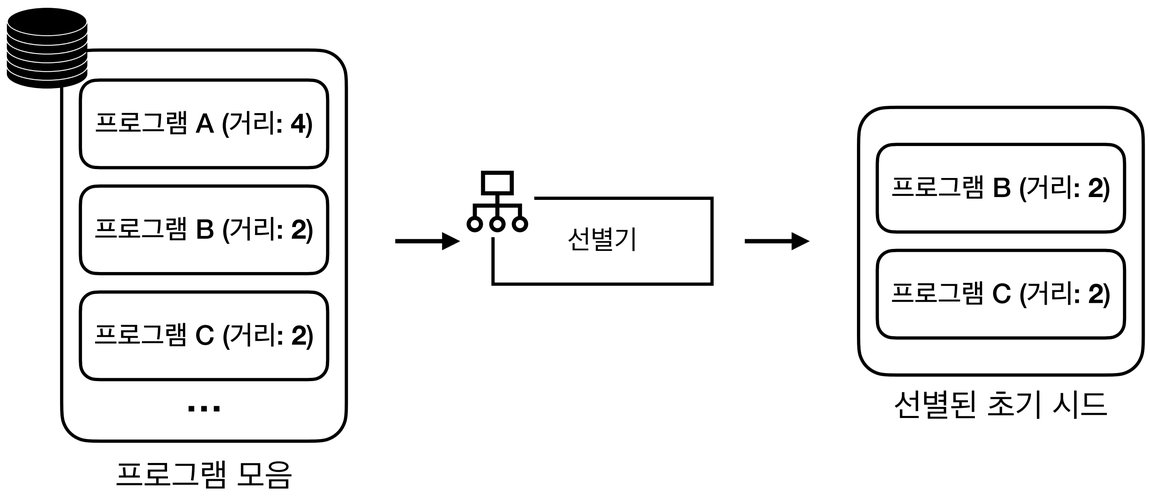
현재 프로그램



컴파일러 소스코드

초기 시드 선별

프로그램 모음에서 목표 최적화와 **가까운** 프로그램들 선별



지향성 퍼징

유효한 프로그램만 생성(변형), 목표 최적화로 변형 유도

- 변이 프로그램의 거리가 가까워지면 시드로 추가, 목표로 유도
- 변이가 목표 최적화를 유발한다면, 번역 검산으로 최적화 검사

목표 최적화: $X \,\%\, 2^C \rightarrow X \,\&\, (2^C - 1)$

```
int foo (int X) {
  return X % 5;
}
```

현재 프로그램 (거리: 1)

→ 변형 →

```
int foo (int X) {
  return X % 4;
}
```

변이 프로그램 (거리: 0)

→ 최적화 →

```
int foo (int X) {
  return X & 3;
}
```

최적화 된 프로그램

번역 검산기

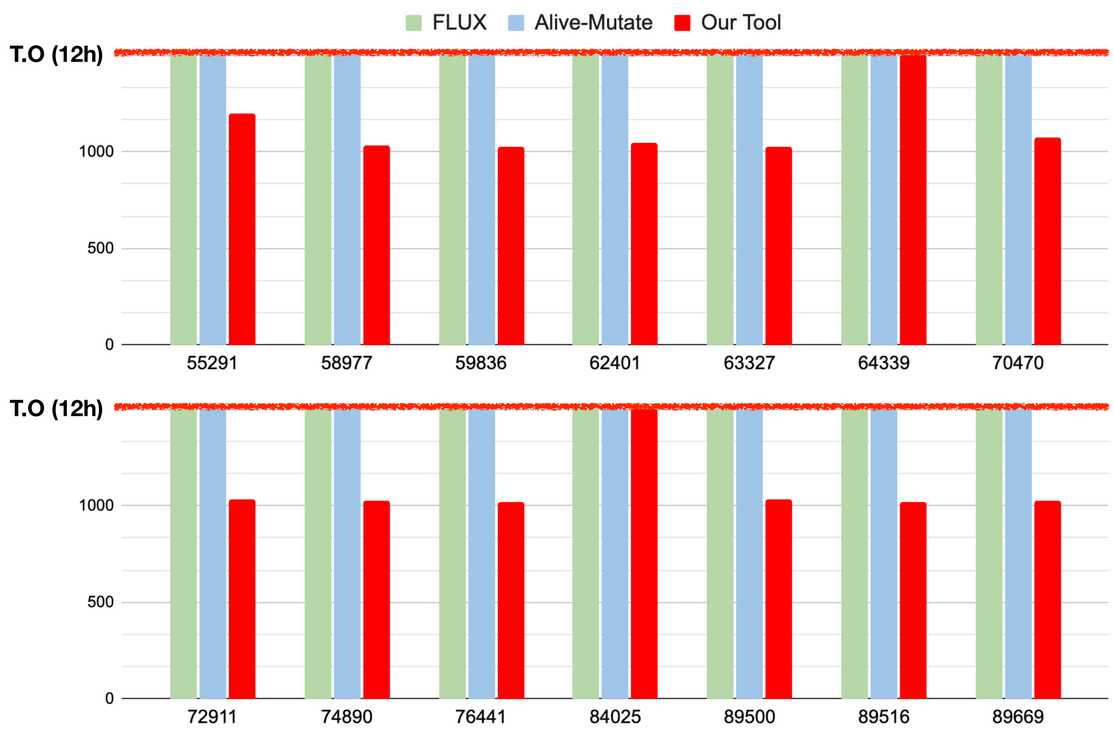
두 프로그램의 의미가 같은가?

실험 및 평가

퍼징 성능 평가

- 실험 대상: 최신 LLVM 최적화 버그 14개
- 실험 목표: 각 버그를 재현하는 프로그램 생성
- 평가 방법: 12시간씩 10번 반복 실험, 각 소요된 시간의 중앙값 비교

14개 중 **12개** 결함을 더 빨리 **20분** 안에 재현!



최신 버전 LLVM 퍼징

- InstCombine 모든 최적화 각각에 대해서 지향성 퍼징 진행 중
- 현재 LLVM 최적화 버그 3개, 번역검산기(Alive2) 버그 1개 발견
- 수정된 번역검산기로 잘못된 최적화 유닛테스트 53개 추가 발견

향후 계획

- 컴파일러 소스코드에서 최적화 조건 힌트를 얻어, 더 똑똑한 변형
- 과거 버그 14종 모두 재현 성공
- 최신 버전 LLVM에서 더 많은 최적화 버그 발견